VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2019 Richard Mička

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATICKÁ DETEKCE POUŽITÉ KRYPTOGRAFIE V KÓDU
AUTOMATIC DETECTION OF CRYPTOGRAPHY USED IN CODE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Richard Mička**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jan Hajný, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**
Ústav telekomunikací

Student: Richard Mička ***ID:*** 195160
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Automatická detekce použité kryptografie v kódu

POKYNY PRO VYPRACOVÁNÍ:

Práce bude zaměřena na analýzu vzorků škodlivého kódu a detekci použitých kryptografických metod pomocí statických či dynamických metod. Vzniknout by měl software umožňující rychlé a automatizované detekování použití symetrického i asymetrického šifrování a dále hashování včetně detailních informací o konkrétním použití (délka klíče, mód atd.). Výsledek by měl být otestován v praxi. V rámci projektu se očekává nastudování celé problematiky a současného stavu řešení, navržení systému, který by vybrané kryptografické algoritmy dokázal detekovat, a dále i jeho plné implementování. Měl by být odevzdána aplikace pracující nad sandboxem zvoleným vedoucím BP. Tento software by měl minimálně zvládat automatickou detekci použití algoritmů z Microsoft CryptoAPI a měl by umět zjištěné informace přehledně prezentovat.

DOPORUČENÁ LITERATURA:

[01] Fast Cryptographic Constants Identification in Retargetable Machine-code Decompilation. [online]. [cit. 2018-09-06]. Dostupné z: http://excel.fit.vutbr.cz/submissions/2015/072/72.pdf.

[02] Eilam, Eldad, and Elliot J. Chikofsky. Reversing : secrets of reverse engineering. Indianapolis, IN: Wiley, 2005. Print.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.
Konzultant: Jakub Křoustek (AVAST Software s.r.o.)

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Táto bakalárska práca sa zaoberá problematikou automatickej detekcie použitej kryptografie zo strojového kódu programu. Analýza použitej kryptografie u neznámej vzorky programu si v súčasnosti vyžaduje veľké manuálne úsilie. V rámci tejto práce bola preskúmaná možnosť automatizácie analýzy využitia kryptografickej knižnice Microsoft CryptoAPI. Táto knižnica je distribuovaná ako súčasť systémov Microsoft Windows a môže byť útočníkom zneužitá na spôsobenie škody obeti. Rozpoznaním operácií s kryptografickými prostriedkami a informovaním o ich využití je možné v niektorých prípadoch zistiť zámer analyzovaného programu a odlíšiť programy so zlým úmyslom len na základe prezentovaného výstupu analýzy. Hlavným cieľom tejto práce bolo vytvorenenie modulu integrovaného do sandboxu Cuckoo na analýzu spôsobu využitia tejto knižnice analyzovanými programami. Behom návrhu tohto analyzátora bola preskúmaná a popísaná funkcionalita CryptoAPI a taktiež nástroj na dynamickú analýzu neznámych programov za účelom rozpoznania malware, sandbox Cuckoo. Navrhnutý analyzátor bol úspešne vytvorený, nasadený, testovaný v praxi a získané výsledky boli následne prediskutované.

KĽÚČOVÉ SLOVÁ

Cuckoo sandbox, dynamická analýza, kryptografia, malware, Microsoft CryptoAPI, ransomware

ABSTRACT

This thesis covers the topic of automatic detection of cryptography used in application code, which currently requires a lot of manual effort to analyze for a given unknown program sample. In this thesis, a possibility of implementing an automated tool for analysing the usage of Microsoft CryptoAPI cryptographic library by analysed programs is researched. This library is distributed with Microsoft Windows and can be misused by an attacker to cause significant harm to a victim. By recognizing cryptographic operations used and by presenting the summary of their use, it is in certain situations possible to distinguish malicious programs just based on the presented analysis summary. Main objective of this thesis was creation of such automatic analyser module integrated into Cuckoo sandbox. Along with the design proposal of such analyser, this thesis includes CryptoAPI library and Cuckoo sandbox functionality exploration and description. Proposed automatic analyser was successfully created, deployed and tested in production environment and the achieved results were discussed.

KEYWORDS

cryptography, Cuckoo sandbox, dynamic analysis, malware, Microsoft CryptoAPI, ransomware

MIČKA, Richard. *Automatická detekce použité kryptografie v kódu*. Brno, 2019, 67 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: doc. Ing. Jan Hajný, Ph.D.

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Automatická detekce použité kryptografie v kódu“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som poďakoval konzultantovi tejto práce Ing. Jakubovi Křoustkovi, Ph.D., za odborné vedenie, konzultácie, cenné podnetné návrhy a podporu pri tvorbe tejto práce. Taktiež by som rád poďakoval vedúcemu práce doc. Ing. Janovi Hajnému, Ph.D. za odborné vedenie a podnety k práci.

Brno ....

podpis autora

Obsah

Úvod **11**

1 Škodlivý kód a metódy jeho analýzy **13**
1.1 Metódy analýzy malware 14
1.1.1 Statická analýza 14
1.1.2 Dynamická analýza 15
1.1.3 Cuckoo sandbox 16

2 Kryptografia na systémoch Windows **24**
2.1 Kryptografické prostriedky 24
2.2 Microsoft CryptoAPI 25
2.3 Asymetrická šifra RSA v CryptoAPI 28

3 Návrh metód detekcie použitej kryptografie v binárnom kóde **29**
3.1 Príprava Cuckoo monitor knižnice 32
3.2 Spracovanie vybraných volaní CryptoAPI 32
3.2.1 Emulácia funkcionality CSP 33
3.2.2 Interpretácia operácií s objektami hešov CryptoAPI 34
3.2.3 Interpretácia operácií s objektami kľúčov CryptoAPI 34

4 Implementácia signatúry v sandboxe Cuckoo **39**
4.1 Architektúra analyzujúcej časti 39
4.2 Prezentácia výsledkov signatúry 41
4.3 Návrh a implementácia testov 42

5 Experimentálne výsledky **44**
5.1 Schopnosti vytvoreného analyzátora 44
5.1.1 Analýza využitia objektov hešov CryptoAPI 44
5.1.2 Analýza využitia objektov kľúčov CryptoAPI 45
5.2 Ukážka schopností analyzátora pri analýze reálneho malware 49
5.3 Štatistiky 51
5.3.1 Záťažový test na dopad na dĺžku analýzy 51
5.3.2 Úspešnosť analýz a spoľahlivosť 52
5.4 Analýza výsledkov získaných automatickými analýzami 53

6 Záver **55**

Literatúra **57**

Zoznam symbolov, veličín a skratiek **60**

Zoznam príloh **61**

A Používané konštanty **62**
A.1 Delenie volaní hešov a kľúčov CryptoAPI 62
A.2 Konštanty operačných módov blokovej šifry 63
A.3 Konštanty používané pri importovaní kľúčov CryptoAPI 63

B Zjednodušený prehľad hookovaných volaní CryptoAPI a zaznamenávaných parametrov **64**

C Dokumentácia priloženého CD **65**
C.1 Obsah CD 65
C.2 Dokumentácia k priloženému obsahu 65
C.2.1 Definície hookov 65
C.2.2 Ukážky výsledkov analýz v HTML 66
C.2.3 Program na testovanie všetkých operácií s CryptoAPI 66
C.2.4 Testovací skript na overenie funkcionalít analyzátora 66
C.2.5 Zdrojový kód analyzátora 66

Zoznam obrázkov

1.1 Funkčné moduly účastné pri analýze v Cuckoo sandbox 18
1.2 Procesy účastné analýzy súboru v Cuckoo sandbox 19
1.3 Signatúry vo webovom prostredí Cuckoo 21
1.4 Rozšírené dáta signatúry vo webovom prostredí Cuckoo 23
1.5 Ukážka hrubého Behavioral logu vo webovom prostredí Cuckoo . . . 23
3.1 Šifrovanie exportovaných použitých kľúčov ďalšími pri ransomware Spora 31
5.1 Ukážka výstupu analýzy pri využití hešov CryptoAPI 45
5.2 Ukážka výstupu analýzy pri využití náhodného kľúča CryptoAPI . . 46
5.3 Ukážka výstupu analýzy pri opätovnom využití známeho kľúča 46
5.4 Ukážka výstupu analýzy pri odvodenom kľúči z výsledku hešu 47
5.5 Ukážka rozpoznávania špeciálnych šifrovaných bufferov 48
5.6 Porovnanie aut. analýzy ransomwaru Alcatraz Locker 50
5.7 Porovnanie aut. analýzy ransomwaru Spora 51

Zoznam tabuliek

5.1 Porovnanie doby behu s analyzátorom a bez neho 52

Zoznam výpisov

1.1 Príklad jednoduchej definície hooku v module `cuckoomon.dll`, ktorá zavolá pôvodnú hookovanú funkciu a zaznamená vstupné a výstupné parametre 20
2.1 Definícia objektu RSAPublicKey v ASN.1 z PKCS#1 [21] 28
3.1 Deklarácia formátu spoločnej hlavičky `BLOBov` 35
3.2 Deklarácia spoločnej hlavičku u kľúčov vo formáte `PUBLICKEYBLOB` a `PRIVATEKEYBLOB` 36
4.1 Definícia štruktúry kontajneru obsahujúceho prezentované výstupné dáta pomocou pseudo-gramatiky 41
A.1 Rozdelenie API volaní na volania týkajúce sa hešov a kľúčo v rámci tejto práce 62
A.2 Konštanty predávané ako parameter pri zmene operačného módu šifry vo volaní `CryptSetKeyParam` 63
A.3 Všetky konštanty používané na identifikáciu typu `BLOBu` 63
B.1 Formát zápisu v nasledujúcich výpisoch 64
B.2 Ukážka transformácie hooku funkcie `CryptCreateHash`, predstaveného vo výpise 1.1, do zjednodušeného formátu definovaného vo výpise B.1. 64

Úvod

V súčasnosti je k Internetu pripojených približne 17,8 miliard zariadení, z ktorých približne 10,8 miliard [1] tvoria zariadenia ako osobné počítače, mobilné telefóny a tablety. Evolúciou zariadení, sú tieto zariadenia schopné prenášať a uskladňovať veľké množstvo informácií, z ktorých niektoré môžu obsahovať citlivé osobné dáta alebo dáta vysokej hodnoty. Tým sa však informácie uchovávané na týchto zariadeniach stávajú zraniteľné a napadnuteľné.

Zvyšovanie objemu zraniteľných dát manipulovaných výpočetnou technikou a zvyšovanie počtu zariadení, potenciálnych obetí, pripojených do siete Internet, priťahuje kyberkriminálnikov [2]. Malware (z angl. *malicious software* – škodlivý software) je software určený na spôsobenie škodlivých úmyslov autora, ktorý je stále viac a viac rozšírený. V súčasnosti, podľa inštitútu AV-TEST sme v roku 2017 evidovali takmer 350 000 nových unikátnych malware denne [3]. Škodu môže malware napáchať viacerými spôsobmi, predovšetkým ukradnutím, poškodením, či zneprístupnením užívateľských dát.

Podstatnú časť v súčasnosti šírených malware tvoria malware, ktorými sa ich autori snažia obohatiť. Na tento účel môžu využívať prostriedky zastrašovania a vyhrážok, ale aj vydieranie. Program po spustení užívateľom zneprístupní dáta užívateľa, niekedy zašifruje obsah užívateľových súborov, a následne požaduje výkupné za opätovné sprístupnenie počítača alebo dát pôvodne obsiahnutých v súboroch. Tento malware býva označovaný ako ransomware (z angl. *ransom* – výkupné) [3].

Ak útočník využije vhodné prostriedky kryptografie na zašifrovanie užívateľských dát, pre obeť sa získanie pôvodných dát z ich zašifrovanej podoby stáva výpočetne nemožné, a jediný spôsob ich opätovného získania je zaplatenie výkupného útočníkovi na vydanie kľúča, pomocou ktorého je následne možné pôvodné dáta zo zašifrovaných súborov získať naspäť. Útočníci týmto spôsobom môžu napáchať značné škody.

Sledovanie aktivity neznámeho programu, z hľadiska spôsobu využitia kryptografických prostriedkov, dokáže prezradiť veľa o neznámom programe. Automatizovaním analýzy využitia prostriedkov kryptografie je možné značne urýchliť a zjednodušiť analýzu neznámej vzorky programu a následné odhalenie jeho prípadných úmyslov spôsobiť škodu užívateľovi. Táto práca bola realizovaná v spolupráci so spoločnosťou Avast.

Preto bol, v rámci tejto práce, rozšírený sandbox Cuckoo, nástroj, používaný spoločnosťou Avast na automatickú analýzu neznámych programov, o rozšírenie, ktoré je schopné automaticky spracovávať aktivitu programu využívajúceho knižnicu Microsoft CryptoAPI. Tieto spracované informácie sú následne prezentované v grafickom prostredí ako súčasť výstupu analýzy analyzovaného programu. Z pre-

zentovanej analýzy je možné jednoducho identifikovať aktivitu programu, ako napríklad použité šifry, vstupné dáta do šifier, použité parametre kľúčov a podobne. Pomocou týchto informácií je analytik schopný rýchlejšie identifikovať úmysly programu. U niektorých z analýz je možné okamžite vyvodiť záver len na základe výstupu analýzy využitých prostriedkov kryptografie, napríklad ak sa jedná o malware, ktorý šifruje užívateľské dáta. Na základe výstupu vytvoreného analyzátora využitej kryptografie je analytik schopný zároveň rýchlo identifikovať spôsob, ktorým sú tieto dáta šifrované.

Táto práca sa venuje výhradne analýze spustiteľných súborov na platformu Microsoft Windows, pretože je to platforma s najvyšším počtom útokov malwaru. Až 67 % všetkých malware útokov v roku 2017 bolo cielených na systém Windows [3]. Taktiež sa zoberá len výhradne analýzou 32-bitových spustiteľných súborov, pretože zo všetkých malware pre systém Windows, tvoria 32-bitové spustiteľné súbory 99,69 % [4].

Práca je rozdelená na 3 celky, kedy sa prvá časť zaoberá malware a spôsobmi jeho analýzy. V tejto kapitole je taktiež opísaný Cuckoo sandbox a jeho prostredie, do ktorého bol vytvorený nástroj na analýzu zakomponovaný. V ďalšej časti je opísaná knižnica CryptoAPI, ktorá môže byť zneužitá a býva využívaná na plnenie plánov útočníkov na spôsobenie škody obeti jeho malware. Opísané sú jej hlavné funkcie, pomocou ktorých je možné vykonávať kryptografické operácie. V nasledujúcej časti je navrhnutý spôsob, ktorým je možné analyzovať využitie tejto knižnice. Ďalej, po návrhu, je opísaný spôsob implementácie analyzátora. Nakoniec sú prezentované schopnosti tohto vytvoreného analyzátora, zhodnotená jeho analýza dvoch vzoriek ransomware z minulosti, zhodnotené metriky výkonnosti a spoľahlivosti tohto vytvoreného rozšírenia a popísané vykonané nadväzujúce experimenty s ním.

1 Škodlivý kód a metódy jeho analýzy

Škodlivé programy, malware, zastrešujú veľké množstvo programov, ktoré je možné deliť na základe viacerých kritérií. Jedným zo spôsobov delenia je napríklad delenie podľa spôsobu šírenia a distribúcie tohto programu. Takto je možné malware rozdeliť na *víry* (viruses), *červy* (worms), *trojské kone* (trojan horses) [5].

Víry sa vyznačujú tým, že sa šíria infikovaním ostatných spustiteľných súborov. Následne, prenesením takto infikovaného programu a spustením na ďalšom počítači, sa vír rozšíri na ďalšie spustiteľné súbory. V súčasnosti víry, závislé na šírení pomocou ľudskej interakcie, upadajú, pretože s popularitou siete Internet rastie počet potenciálnych obetí pre ďalší typ malwaru.

Ním sú *červy*, ktoré sa šíria samostatne, narozdiel od vírov, s využitím siete. Sú tak nezávislé od interakcie s ľuďmi. Šíriť sa môžu viacerými spôsobmi, napríklad využívaním zraniteľností operačných systémov, či programov používaných na systémoch obetí.

Posledným typom malwaru podľa spôsobu šírenia je *trojský kôň*. Ten sa vyznačuje šírením ako súčasť iného programu, ktorý zdanlivo vyzerá neškodne, síce v sebe ukrýva škodlivý kód. Program môže fungovať správne, ako program, za ktorý sa vydáva, a v pozadí môže vykonávať škodlivú aktivitu. Taktiež sa takto označujú súbory, ktoré spoliehajú na oklamanie užívateľa tak, že sú prezentované ako iný nevinný súbor, napríklad multimediálny súbor.

Malware je možné kategorizovať podľa následkov jeho infekcie, či jeho účelu. Malware môže byť tvorený za účelom získania prístupu do cudzieho systému, vtedy sa vraví o *zadných vrátach* (backdoor access). Pomocou tohto typu malwaru môže útočník získať až úplnú kontrolu nad systémom obete. Taktiež sa dá kategorizovať malware, ktorý sa snaží len spôsobiť škodu obeti formou vandalizmu, či ukradnúť informácie a dáta. Ďalší z odlišných spôsobov chovania škodlivého programu je malware zameraný na získanie finančného zisku vydieraním [6]. Ako ransomware môžeme označiť kategóriu software, ktorý po spustení obmedzí funkcionalitu počítača. Následne zobrazí správu o požadovanom výkupnom. Tento malware však nemusí spôsobiť škodu a užívateľ môže po odstránení tohto ransomware pokračovať vo využívaní jeho počítača bez žiadnych následkov.

Na donútenie obetí zaplatiť výkupné začali autori ransomwaru používať prostriedky kryptografie, kedy program po spustení zašifruje užívateľove súbory s tým, že ich drží ako zálohu či rukojemníka na vynútenie zaplatenia výkupného. Táto podskupina ransomwaru býva niekedy označovaná aj ako *filecryptor* alebo *cryptoware* [7].

1.1 Metódy analýzy malware

K analýze neznámeho programu sa dá pristupovať dvomi základnými rozdielnymi spôsobmi, statickou analýzou a dynamickou analýzou. Každý z týchto spôsobov analýzy je viac vhodný na extrahovanie rozdielnych informácií, ktoré po pospájaní umožnia analytikovi získať presnejšiu predstavu o funkcionalite analyzovaného programu. Avšak pre každý z týchto spôsobov analýzy existujú spôsoby, ktorými môžu autori malware sťažiť a skomplikovať analýzu. Každý program je však náchylný na analýzu, pretože všetka jeho funkcionalita, na to aby mohol program úspešne fungovať, musí byť obsiahnutá v jeho strojovom kóde. Neexistuje dokonalý spôsob, ktorým by bol autor schopný úplne ukryť funkcionalitu svojho programu, pretože vždy program musí byť spustiteľný a realizovateľný procesorom. V následujúcich podčastiach sú popísané rozdiely medzi statickou a dynamickou analýzou, používané a známe nástroje na ich vykonávanie, a tiež aké informácie je možné získať s použitím daných nástrojov.

1.1.1 Statická analýza

Pri statickej analýze je spustiteľný súbor analyzovaný bez jeho spustenia. Statickou analýzou je možné získať informácie o jeho kóde a dátach, ale aj o formáte samotného spustiteľného súbory, ktorým je daný program reprezentovaný. Analýza je vykonávaná na základe výstupu z nástrojov, ktoré sú buďto schopné z binárneho kódu vytvoriť pre ľudí jednoduchšiu reprezentáciu na pochopenie, alebo vedia zistiť informácie z informatívnych častí (napr. hlavička) spustiteľného súboru, ktoré následne prezentujú v pre ľudí pochopiteľnej podobe [5].

Na statickú analýzu sú používané takzvané *disassemblery*. Sú to programy, ktoré spätne konvertujú binárny strojový kód na strojové inštrukcie špecifické pre danú architektúru, reprezentované mnemotechnickými skratkami. Výsledný kód následne prezentujú ako súvislý výpis inštrukcií, sekvenčne v poradí, v akom sa nachádzajú v programe, alebo ako graf toku programu (control flow graph – CFG). CFG reprezentuje súvislosti medzi základnými blokmi kódu, reprezentované vrcholmi tohto grafu, pomocou šipiek zobrazujúcimi tok pri podmienečne vykonávaných skokoch medzi blokmi [9].

Pri rekonštrukcii jazyku symbolických adries disassembler naráža na problém s odlíšením kódu a dát. Ak disassembler prekladá na princípe *linear sweep* (lineárny prechod), prekladá sekcie určené pre kód postupne ako plynú za sebou. Nedostatkom tohto postupu je, že niekedy sa v kóde môžu objaviť časti dát, pri ktorých by disassembler nebol schopný preložiť danú sekvenciu, pretože by sa ju snažil preložiť ako inštrukcie. Druhý postup je *recursive traversal* (rekurzívny priechod), kedy sa

disassembler snaží nasledovať cestu kódu a vnoruje sa pri podmienečnom vetvení [8].

Ďalším nástrojom na statickú analýzu je *decompiler*, ktorý prekladá binárny strojový kód na jazyk vyššej úrovne, ktorý môže v niektorých situáciach byť ľahší a intuitívnejší na pochopenie. V oboch prípadoch decompilera a disassemblera je nutné ich výstup manuálne analyzovať. Táto analýza môže byť skomplikovaná v prípade použitia anti-reverzovacích praktík na zmätenie disassemblerov a decompilerov, napríklad použitím *packeru*[1], či obfuskačných techník, ako transformáciu kódu na funkčne ekvivalentný kód, ktorý je ale netradičnou postupnosťou inštrukcií pre človeka ťažší na pochopenie.

1.1.2 Dynamická analýza

Dynamická analýza využíva nutnú nedokonalosť a neodolnosť programu na analýzu (sekcia 1.1). Pri tejto analýze sa program spúšťa a jeho aktivita je sledovaná. Je tak možné odhaliť pravé správanie programu, pretože na to, aby program niečo spôsobil, je nútený využívať systémové knižnice poskytujúce rozhranie na systémové volania. Pomocou tejto analýzy je možné sa dozvedieť omnoho viac informácií ako len pri samotnej statickej analýze výstupu disassembleru, pretože program je schopný sa za behu meniť – tieto zmeny sa pri disassemblovaní bez spustenia neprejavujú. Na monitorovanie aktivity programu za jeho behu slúži predovšetkým *debugger*. Debugger zobrazuje dáta v pamäti a registroch v stave pri aktuálnom bode behu programu a umožňuje sledovať ako je s nimi manipulované pri jednotlivých inštrukciách, vykonávanie ktorých je možné manuálne ovládať. Debugger je použiteľný pri manuálnej analýze, ale neposkytuje prostriedky na efektívnu automatickú analýzu [5].

Ďalším prostriedkom na dynamickú analýzu je *sandbox*, ktorý poskytuje plnú kontrolu nad programom. Program spustený v sandboxe by mal byť schopný vykonávať všetky úkony akoby bol bežne spustený na bežnom systéme, bez možnosti vplyvu na prostredie mimo sandbox. Prostredie sandboxu môže byť emulované alebo súčasť plne virtualizovaného prostredia. V prípade virtualizovaného prostredia sandbox využíva *hooky*. Hookovanie je proces, pri ktorom je prepísané pôvodné volanie funkcie na volanie inej funkcie. Je tak možné presmerovať pôvodné volanie do vlastného kódu, kde je možné narábať so získaným tokom programu. Pomocou hookov je tak možné transparentne pre bežiaci program zbierať dáta o jeho volaniach a parametroch v nich použitých.

Tak ako existujú spôsoby na sťaženie statickej analýzy, aj dynamická analýza je náchylná na techniky určené na jej skomplikovanie. Existuje veľa takzvaných *anti-debugging* či *anti-sandbox* techník zameraných na detekciu prostredia debuggeru či

[1] Je to program, ktorý zabalí, prípadne skomprimuje, či zašifruje zdrojový kód programu tak, že sa pri spustení rozbalí bez pozmenenej funkčnosti.

sandboxu, pomocou ktorých by program pozmenil svoju činnosť, aby nebola jeho pravá funkcionalita vykonaná v debuggeri či sandboxe. Taktiež je možné obchádzať analýzu v sandboxe oneskorením aktivity programu o niekoľko minút. Analýza v sandboxe trvá rádovo tiež len niekoľko minút a každá minúta naviac je viac nákladná, pretože požaduje dlhšie obsadenie zdrojov používaných na analýzu, v tomto prípade virtuálnych strojov. Dobrý sandbox by tak mal byť schopný ukryť svoju prítomnosť, detegovať chovanie programu pokúšajúceho zistiť prítomnosť sandboxu, či transparentne skracovať čakanie, pomocou ktorého by program oddialil svoju aktivitu.

1.1.3 Cuckoo sandbox

Cuckoo sandbox je sandbox vyvíjaný ako open-source pod licenciou GNU General Public License, verzia 3. V súčasnosti sa dá rozdeliť na dve vetvy. Prvá, pôvodná vetva projektu Cuckoo sandbox, vedená v súčasnosti organizáciou *Cuckoo Foundation*, s najnovšou verziou 2.0.6 [10]. Ako druhú je možné vyčleniť vetvu, ktorá sa sama označuje ako *cukoo-modified*, ktorej vývoj bol v roku 2014 osamostatnený a ďalej vedený firmou Optiv, Inc. [11]. Oproti vtedy aktuálnej verzii Cuckoo 1.3 bola rozšírená o viac ako 250 zmien, ktorými ponúkala vyššiu presnosť analýz ako hlavný projekt Cuckoo [13]. Vývoj tejto vetvy pokračoval komunitou, avšak tento vývoj je vo verejnej podobe zastavený, finálna verzia je dostupná na repositári Github [12] s poslednou zmenou 26. 4. 2017. V rámci tejto práce bude vždy sandboxom Cuckoo myslené pokračovanie tejto verzie, ktorej vývoj interne pokračuje a ktorá je interne používaná.

Tento sandbox je nástroj na spúšťanie a analýzu súborov pre rôzne platformy, či už systémy Microsoft Windows, alebo systémy s jadrom Linux, s architektúrou `x86` alebo `amd64`. Tieto súbory sú spustené v izolovanom prostredí a ich aktivita počas behu je podrobne sledovaná. Sledované sú volania funkcií, vytváranie procesov, súbory, s ktorými program počas behu narába, jeho sieťová aktivita, ktorú zachytáva do formátu PCAP. Takto je možné analyzovať nielen výhradne spustiteľné súbory. Také súbory, ktoré nie sú priamo spustiteľné na danej platforme, sú spúšťané v programoch schopných ich spustiť a sú pomocou nich analyzované. Medzi tieto podporované formáty patria napríklad súbory dynamicky linkovaných knižníc DLL, ktoré sú spúšťané vstavaným nástrojom systémov Windows `RunDLL32.exe`, súbory jazyku Python, ktoré sú spúšťané v príslušnom interpretéri jazyku Python, dokumenty Microsoft Office, ktoré sú spúšťané v Microsoft Office programoch, URL a HTML súbory, ktoré sú analyzované spustením vo webovom prehliadači. Základným výstupom analýzy v Cuckoo je záznam vykonaných volaní API, záznam sieťovej aktivity PCAP, detegované signatúry chovania, hodnotenie na základe závažnosti detegova-

ných signatúr nazvané „malscore“, výsledok vyhodnotenia porovnaniu s pravidlami v jazyku YARA, zhrnutie aktivity so súbormi.

Nasledujúce informácie čerpajú z dokumentácie projektu Cuckoo [12], alebo sú výsledkom analýzy kódu sandboxu Cuckoo autorom tejto práce za účelom implementácie a integrácie produktu tejto práce.

Architektúra sandboxu

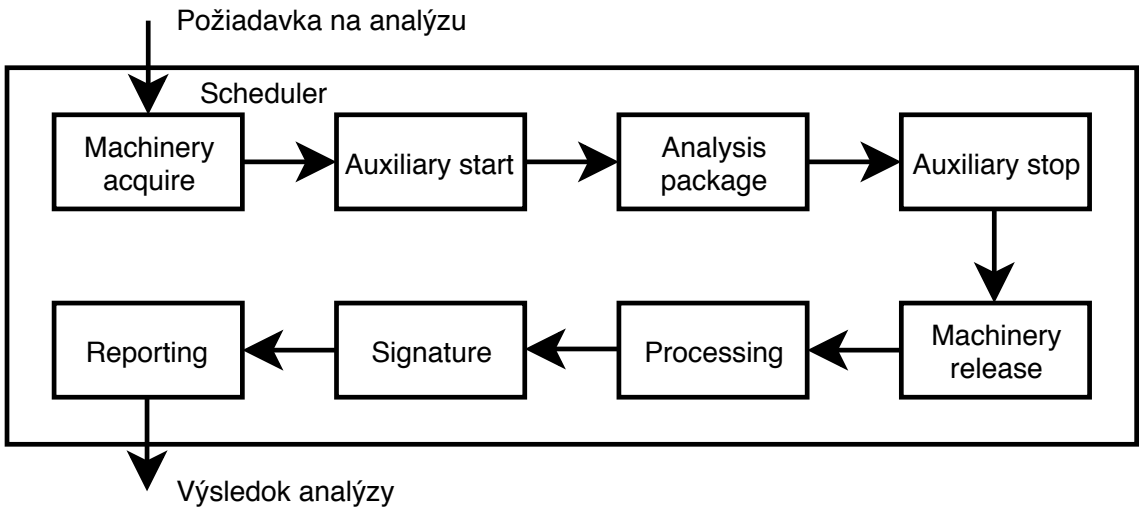
Za vrchnú vrstvu sandboxu Cuckoo je možné považovať jeho webové rozhranie tvorené frameworkom *Django* napísaným v jazyku Python 2. Pomocou tohto rozhrania je možné spúšťať analýzy, či už manuálne pomocou HTML rozhrania, alebo pomocou HTTP REST API, ktoré umožňuje jednoduchú interoperabilitu s ostatnými systémami. Webové rozhranie taktiež tvorí jednu z možností na získanie a zobrazenie výsledku analýzy.

Cuckoo sandbox je navrhnutý s ohľadom na modulárnosť a jednoduchú rozšíriteľnosť. Ponúka tak možnosti na rozširovanie a vytváranie vlastných modulov, či už spravujúcich nový typ analýzy, alebo spracovanie výsledkov. Tieto moduly sú rozdelené na nasledovné kategórie:

- Machinery moduly – moduly spravujúce stroje používané na analýzu, pomocou ich rozhrania je Cuckoo nezávislé na použitých prostrediach na analýzu, umožňujú využitie či izolovaných fyzických strojov, tak aj virtualizovaných prostredí na analýzu,
- Auxiliary moduly – reprezentujú moduly, ktoré sú spúšťané pri každej analýze pred spustením analyzovaného súboru a následne po ukončení behu analyzovaného súboru. Sú flexibilné a schopné plniť ľubovoľné funkcie,
- Analysis package – je to balík zastrešujúci viaceré typy analýz, napríklad všetky pre stroje s určitým operačným systémom, ktorý vykonáva všetky úkony počas analýzy súboru,
- Processing moduly – spracovávajú hrubé nespracované výsledky extrahované z analýzy na dáta ľahšie použiteľné ďalšími etapami v spracovaní analýzy,
- Signature moduly – analyzujú spracované dáta a analyzujú ich za účelom identifikácie konkrétneho chovania, ktoré majú detegovať a následne naň upozorniť,
- Reporting moduly – sú posledné moduly v reťazci analýzy a slúžia na vytvorenie výstupu analýzy ako napríklad vo vlastnom formáte, formáte JSON, alebo ako databázových operácií.

V obrázku 1.1 je možné vidieť ako sú vyššie opísané moduly zoradené pri analýze.

Po odoslaní požiadavky na analýzu súboru sa spustí nastavený *machinery modul*, ktorý pripravý stroj na analýzu. Na tomto stroji sa spustí v Python interpretéri Python modul Cuckoo `agent.py` spravujúci systém a spojí sa so spravujúcou in-



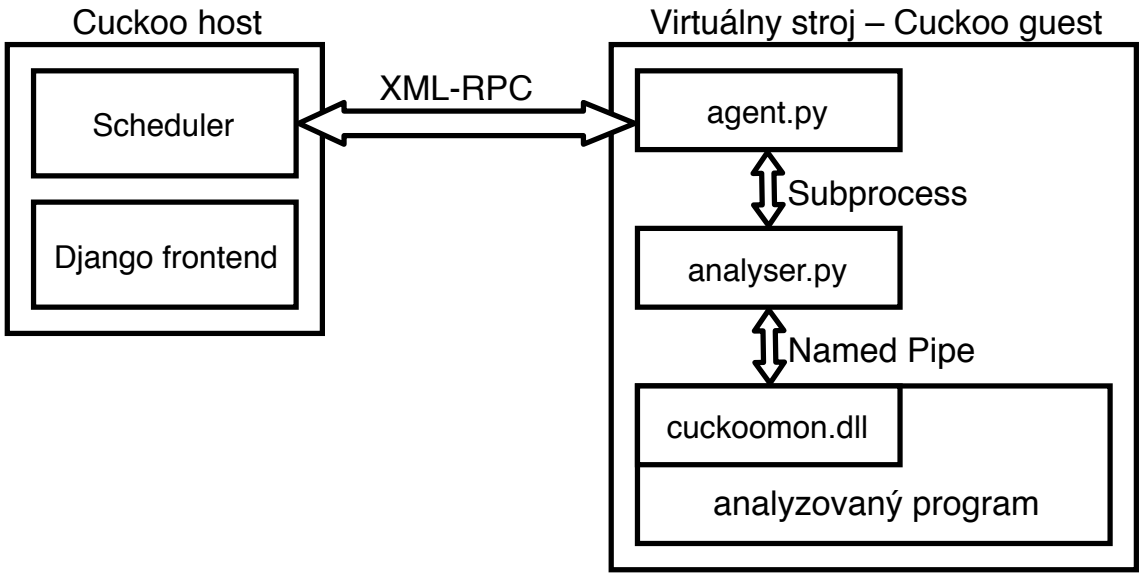
Obr. 1.1: Reťazec modulov zúčastňujúcich sa pri analýze

štanciou Cuckoo (taktiež označovaná ďalej ako *host*). Host následne nahraje na stroj použitý (ďalej označovaný ako *guest*) analyzačný balík, ktorého jadro tvorí modul `analyzer.py`. Tento modul spúšťa analyzovaný program a zberá z neho dáta o jeho chovaní, zároveň však ukrýva pre analyzovaný súbor svoju prítomnosť. Následne, po skončení analýzy, tento modul `analyzer.py` kontaktuje modul `agent.py`, ktorý získané dáta predá Cuckoo *hostovi* na ďalšie kroky analýzy nasledujúce po analyzačnom balíku `Analysis package`.

Extrakcia aktivity analyzovaného procesu

O sledovanie chovania bežiaceho analyzovaného programu sa stará `analyzer.py`, ktorý využíva hookovanie volaní funkcií z dynamicky linkovaných knižníc (ďalej ako *DLL* – z angl. *Dynamic-link library*) pomocou knižnice Cuckoo monitor, implementovanou v module `cuckoomon.dll`. Analyzovaný program je spustený ako pozastavený a pred jeho pokračovaním je doň injektovaná `cuckoomon.dll`. Injektovanie DLL je metóda, kedy je do pamäte procesu pridaná knižnica, ktorú samotný program nedeklaroval ako potrebnú v tabuľke importovaných knižníc a ani ju sám nenahral. Existuje viac spôsobov na realizáciu tohto efektu, v Cuckoo sa využíva injekcia pomocou volaní APC `QueueUserAPC` alebo spustením dočasného vlákna v analyzovanom programe pomocou funkcie `CreateRemoteThread`. Knižnica `cuckoomon.dll` následne s využitím objektov *Named Pipe* komunikuje s modulom `analyzer.py`, ktorý ovláda jej aktivitu.

Pri nahraní modulu `cuckoomon.dll` do sledovaného procesu sa spustí jeho inicializačná funkcia, ktorá nastaví všetky deklarované hooky. Definície hookov sa na-



Obr. 1.2: Ukážka architektúry Cuckoo z hľadiska stroja použitého na analýzu a bežiacich procesov a spôsobov ich vzájomnej komunikácie

chádzajú v hlavnom súbore modulu `cuckoomon.c` v poli `g_hooks` typu `hook_t`. Pri nahraní `cuckoomon.dll` do procesu sa inicializujú všetky deklarované hooky práve v tomto poli `g_hooks`. Definícia hooku pozostáva z názvu DLL a názvu jej exportovanej funkcie, ktorú chceme hookovať. Cuckoomon sa následne stará o sledovanie, či daná knižnica DLL nie je nahraná v procese. Ak deteguje prítomnosť či nahranie sledovanej knižnice, postará sa o hookovanie všetkých jej sledovaných exportovaných funkcií.

Následne sú všetky volania danej funkcie presmerované do užívateľových vlastných definícií hookov. V rámci implementácie hooku je možné zaznamenávať vstupné, či výstupné parametre funkcie. Je taktiež možné vykonanie danej funkcie úplne vynechať. Cuckoo monitor transparentne poskytuje ochranu a krytie vlastných nepôvodných volaní, napríklad zachytávaním chybových kódov pri volaní hookovanej funkcie a následne obnovením tohto kódu, aby neprišlo k jeho nechcenej výmene, ktorú by mohol analyzovaný program zle vyhodnotiť. Vo výpise 1.1 je ukázaná definíciu hooku na funkciu `CryptCreateHash`, v ktorej sú pomocou makra `LOQ_bool` zaznamenané vstupné a výstupné parametre funkcie. Ostatné hooky sú k dispozícii na priloženom CD (príloha C) vo formáte popísanom v prílohe B. Každé takto zaznamenané volanie je zbierané vo formáte BSON (Binary JSON) s vlastnou schémou špecifickou pre Cuckoo. Všetky súbory zberajúce volania hookovaných funkcií knižníc vo formáte BSON sú po skončení behu programu pozbierané a predané spolu s ďalšími súbormi analýzy *hostovi* Cuckoo, na ktorom sú spracované a analyzované.

Výpis 1.1: Príklad jednoduchej definície hooku v module `cuckoomon.dll`, ktorá zavolá pôvodnú hookovanú funkciu a zaznamená vstupné a výstupné parametre

```
HOOKDEF(BOOL, WINAPI, CryptCreateHash,
  _In_   HCRYPTPROV hProv,
  _In_   ALG_ID Algid,
  _In_   HCRYPTKEY hKey,
  _In_   DWORD dwFlags,
  _Out_  HCRYPTHASH *phHash
) {
  BOOL ret = Old_CryptCreateHash(hProv, Algid, hKey, dwFlags,
    ↪ phHash);
  LOQ_bool("crypto", "phpiP", "hProv", hProv, "Algid", Algid,
    ↪   "hKey", hKey, "Flags", dwFlags, "hHash", phHash);
  // v druhom parametre makra LOQ_bool
  // je špecifikovaný formát
  // 'p' a 'h' znamená 32 bitové ukazovatele
  // 'i' znamená 32 bitové celé číslo
  // 'P' znamená ukazovateľ na ukazovateľ
  // za tým nasledujú dvojice názov parametru,
  // pod ktorým bude dostupný v Cuckoo a jeho hodnota
  return ret;
}
```

Spracovanie výsledkov analýzy

Na spracovanie surových záznamov chovania analyzovaného programu slúžia moduly *Processing* a následne *Signature*. Najprv sú výstupné súbory analýzy predané modulu *Processing*, ktorý analyzuje priamo dáta z behu programu v binárnej podobe. Hlavným modulom *Processing* je modul `behavior.py`, ktorý vie analyzovať záznamy hookovaných API volaní knižníc z formátu BSON na formáty ľahko spracovateľné v jazyku Python. Zo záznamov volaní pre jednotlivé procesy vytvorí Python štruktúru `list`, pozostávajúcu z jednotlivých záznamov volaní, reprezentovaných v Python kontajneri `dictionary`, v ktorom sa jednotlivé informácie o volaní dajú získať pomocou kľúčových hodnôt, ktorými sú spravidla Python reťazce `string` či `unicode`.

Takto spracované binárne dáta z analýzy do natívnych datatypov jazyku Python sú následne predávané modulom *Signature*. Modul *Signature* je tvorený Python triedami dediacich z abstraktnej triedy `Signature`, ktorej definícia je súčasťou vnútornej knižnice Cuckoo. Signatúram je v Cuckoo umožnený prístup k dátam z *Processingu* pomocou troch metód triedy, ktoré musia definovať:

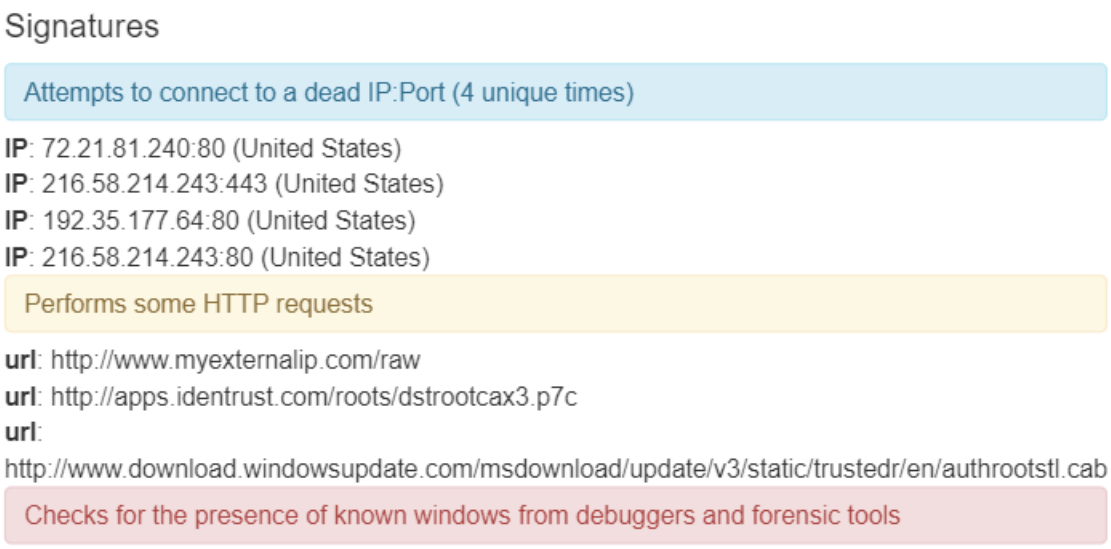
- metóda `run`,
- metóda `on_complete`,

- metóda `on_call`.

Metódy `run` a `on_complete` sú volané len jedenkrát ako posledné metódy v module *Signature*, preto sú oproti využitiu metódy `on_call` neefektívne. Metóda využitia `on_call` funguje tak, že signatúra deklaruje, ktoré volania chce sledovať. Následne je pre dané volania vždy volaná metóda `on_call` v rámci jedného prechodu histórie volaní. Výhodou je, že metódy `on_call` sú spúšťané paralelne pre všetky signatúry z modulu *Signature*, to znamená, že pri dlhom zázname aktivity sú ušetrené zbytočné opakované iterácie dlhých zoznamov. Signatúra následne môže vrátením hodnoty `True` prehlásiť, že bola detegovaná a tým sa pridá do zoznamu detegovaných signatúr, ktorých výstup je následne prezentovaný.

Prezentácia výsledkov analýzy

Súčasťou definície signatúry je jej názov `name` a popis `description`. Ak signatúra signalizuje, že bola detegovaná, jej názov sa objaví vo výstupe detegovaných signatúr. To spôsobí jej zobrazenie vo výstupe analýzy, kde sa zobrazí jej popis. Signatúra má tiež k dispozícii možnosť, okrem popisu, poskytovať jej ďalšie informácie popisujúce dôvod jej detekcie. Tieto informácie môže predať v kontajneri `data` alebo `new_data`, ktoré sú zobrazované webovým prostredím po kliknutí na detegovanú signatúru.

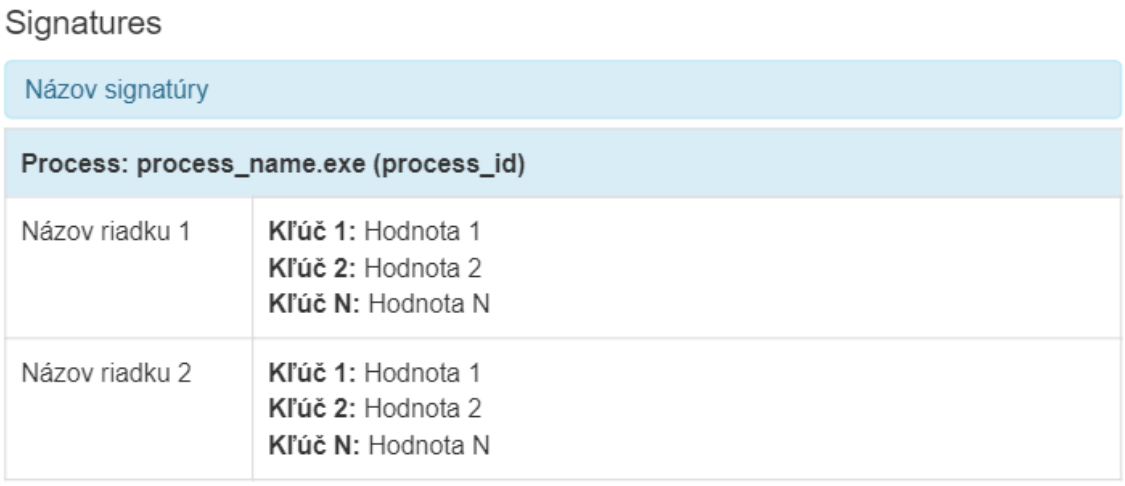


Obr. 1.3: Ukážka reprezentácie detegovanej signatúry vo webovom prostredí výsledku analýzy.

Ak signatúra použije na rozširujúce informácie kontajner `data`, jeho obsah je vykreslený ako zoznam párov „**kľúč**: hodnota“ (príklad tohto zobrazenia je vidno v obrázku 1.3). Kontajner `new_data` poskytuje sofistikovanejší spôsob prezentácie dát signatúry pomocou HTML tabuľky, ktorá sa po rozkliknutí signatúry ukáže. Jej podoba je ukázaná v obrázku 1.4. Potrebná štruktúra tohto kontajneru pre správne vykreslenie tabuľky je podrobne popísaná v sekcii 4.2.

Okrem signatúr, Cuckoo poskytuje veľa ďalších spôsobov reprezentácie získaných dát z analýzy, ako je napríklad *Behavioral log*, kde je možné ručne prezrerať záznamy hookovaných API volaní. Jeho podoba je na obrázku 1.5.

Taktiež je možné prezentovať výsledky analýzy použitím modulov *Reporting*. Pomocou týchto modulov je možné si vytvoriť reprezentáciu výsledku vo vlastnom formáte, alebo prepojiť systém Cuckoo s iným vlastným systémom. Jedným z hlavných modulov *Reporting* je, napríklad, modul na vytváranie výsledkov analýzy v štandardnom formáte JSON (JavaScript Object Notation).



Obr. 1.4: Ukážka rozšírených dát signatúry v tabuľke.

Obr. 1.5: Ukážka *Behavioral logu* analýzy súboru, kde je vidieť výber zo záznamu hookovaných API volaní a detail jedného volania `NtCreateFile` do systémovej knižnice `ntdll.dll`.

2 Kryptografia na systémoch Windows

V tejto časti práce je najprv popísaná terminológia z oblasti kryptografie používaná v tejto práci. Následne je popísaná knižnica Microsoft CryptoAPI a jej architektúra a funkcionalita. V závere kapitoly je popísaná asymetrická šifra RSA a reprezentácia jej kľúčov v CryptoAPI podľa štandardu PKCS#1.

2.1 Kryptografické prostriedky

Kryptografické funkcie je možné rozdeliť na 3 hlavné kategórie [14]:

- symetrické šifry,
- asymetrické šifry,
- hešovacie funckie.

Symetrické šifry sa vyznačujú tým, že sú to šifry, ktoré používajú na šifrovanie a dešifrovanie rovnaký kľúč k, takže sa dajú charakterizovať ako funkcie, pre ktoré platí: $c = f(m, k)$, kde m otvorený text a c je výstup funkcie reprezentujúci šifrový text. Na dešifrovanie sa používa funkcia inverzná k f, ktorá má ako parameter kľúča rovnaký kľúč k, dá sa teda opísať ako: $m = f^{-1}(c, k)$. Niekedy býva kľúč symetrickej šifry označovaný ako tajný kľúč, pretože musí byť uchovávaný v tajnosti. Vďaka symetrii, ktokoľvek disponujúci kľúčom k môže dešifrovať šifrový text a aj šifrovať otvorený text.

Asymetrické šifry sa od symetrických líšia tým, že využívajú na šifrovanie a dešifrovanie rozdielne kľúče. V našom príklade bude d reprezentovať dešifrovací kľúč a e šifrovací kľúč. Platí teda: $c = f(m, e)$, kde m reprezentuje otvorený text a c je šifrový text. Na dešifrovanie šifrovéhu textu vytvoreného algoritmom asymetrickej šifry je potrebný dešifrovací kľúč d a pomocou šifrovacieho kľúča e nie je možné spätne získať zo šifrového textu c pôvodný otvorený text m. Dešifrovanie sa dá opísať ako: $m = f^{-1}(c, d)$. Hlavnou výhodou asymetrickej šifry je, že na zašifrovanie dát nie je vôbec potrebný dešifrovací kľúč d, ten môže mať príjemca bezpečne uchovaný u seba, čo robí šifru vhodnou na použitie pri prenose správ po nezabezpečenom kanále.

Hešovacie funckie sú funkcie, ktoré z voliteľne veľkého vstupu m generujú výstup konštantnej dĺžky h, zvaný heš, hešový kód, odtlačok, či anglicky ako hash, hash code, fingerprint. Dĺžka výstupu h môže mať rôzne veľkosti, napríklad 128 bitov, 160 bitov, či 256 bitov, ktoré sú charakteristické pre danú hešovaciu funckiu. Hešovacie funkcie sa taktiež vyznačujú ďalšími vlastnosťami, jednocestnosťou a bezkoliznosťou. Jednocestnosť hešovacej funkcie znamená, že pre danú správu m, je jednoduché spočítať h ako $h = H(m)$, kde H predstavuje konkrétnu hešovaciu funkciu. Avšak, ak je známy výstup hešovacej funkcie h, je veľmi obtiažne vypočítať pôvodnú správu m. Druhá vlastnosť hešovacej funkcie je bezkoliznosť, ktorá

znamená, že je veľmi obtiažne nájsť dve rôzne správy m a $m\prime$, pre ktoré by platilo $H(m) = H(m\prime)$. Hešovacie funkcie sú používané na generovanie jednoznačných identifikátorov dát, čo je uplatniteľné napríklad pri digitálnych podpisoch, kontrole integrity. Tiež je možné hešovacie funkcie uplatniť v pseudonáhodných generátoroch.

2.2 Microsoft CryptoAPI

Kryptografické funkcie sú na platformách Microsoft Windows, od vydania operačného systému Windows NT 4.0, k dispozícii v knižnici Cryptographic Application Programming Interface, skôr známou pod názvom Microsoft CryptoAPI. Predstavuje ju sústava dynamicky linkovaných knižníc, ktoré predstavujú abstrahované rozhranie na vykonávanie operácií využívajúcich kryptografické prostriedky. Kryptografické prostriedky sú zhromažďované a implementované v takzvaných cryptographic service providers (CSP), sprostredkovateľoch kryptografických služieb. Tieto CSP tiež spravujú a uchovávajú kryptografické kľúče. Systém Windows je distribuovaný s viacerými CSP a taktiež je možné pridávanie vlastných modulov CSP [15].

Moduly CSP sú navrhnuté tak, že aplikácia nešpecifikuje všetky detaily kryptografických operácií. Rozhranie CSP umožňuje aplikácii voľbu použitého šifrovacieho či iného algoritmu, ale implementácia algoritmu sa nachádza vo vnútri CSP, ku ktorému aplikácia prístup nemá. Taktiež má aplikácia využívajúca CryptoAPI zamedzený prístup k dátam kľúča, ktoré su priamo použité k šifrovaniu. Pridáva to tak vrstvu bezpečnosti pre aplikácie tým, že sú nútené generovať kľúče v moduloch CSP. Ku kľúčom knižnice CryptoAPI je následne nutné pristupovať pomocou neurčitých objektov *handle*[1], reprezentujúcich objekty kľúčov a hešov pomocou unikátnych 32-bitových celočíselných hodnôt.

Aplikácia, ktorá chce využívať knižnicu CryptoAPI musí ako prvé načítať dynamicky linkovanú knižnicu sprostredkujúcu funkcie CryptoAPI a získať potrebné adresy funkcií, ktoré chce následne používať. Na to, aby mohla aplikácia začať vykonávať operácie s kryptografickými primitívami, je potrebné získať takzvaný kontext CSP, ktorý je reprezentovaný 32-bitovým unikátnym *handlom*. Na získanie handlu je potrebné špecifikovať CSP a taktiež musí byť špecifikovaný názov kontajneru na ukladanie kľúčov, takzvaný *key container*. Každý *key container* je schopný uchovávať dva asymetrické kľúčové páry, jeden použiteľný len na overovanie a vytváranie digitálnych podpisov, označovaný aj ako `AT_SIGNATURE`. Druhý z nich slúži na ustanovenie symetrického kľúča alebo taktiež priame šifrovanie ľubovoľných dát, označovaný ako `AT_KEYEXCHANGE`. Následne je možné vytvárať objekty CryptoAPI

[1] Je to unikátna číselná hodnota priradená konkrétnemu internému objektu, ku ktorému nemá aplikácia zvonku prístup. Pomocou tejto číselnej hodnoty, ktorá je následne predávaná knižnici ako parameter operácií, je knižnica schopná identifikovať objekt, s ktorým má byť operácia prevedená.

pomocou funkcií na ich vytváranie s použitím získaného handlu kontextu CSP. Po získaní handlu samostatného objektu je možné s ním manipulovať len na základe prideleného handlu zo strany CryptoAPI. Asymetrické kľúče sú uchovávané v príslušnom kontajneri, v ktorom boli vytvorené aj po uvoľnení kontextu CSP a je možné k nim znova pristúpiť otvorením nového CSP kontextu s rovnakým názvom kontajneru. Kľúče symetrických algoritmov však nie je možné ukladať v kontajneri a ich hodnoty sa strácajú pri uvoľnení kontextu CSP. Taktiež je možné vytvárať dočasné kontexty CSP, kedy je vytvorený dočasný prázdny kontajner kľúčov, vtedy nie sú uchovávané ani asymetrické kľúče a ich hodnoty sú rovnako zahodené pri uvoľnení kontextu CSP.

Knižnica CryptoAPI je označená Microsoftom, ako zastaralá, a od verzie Windows Vista je v systémoch distribuovaný nástupca CryptoAPI, *Cryptography API: Next Generation (CNG)* [16]. CryptoAPI je však stále distribuovaná aj s v súčasnosti najnovším vydaním systému Windows a podporuje systémy staršie ako Windows Vista, ako Windows XP.

Následkom toho, že je CryptoAPI už zastaralá, neposkytuje veľa v súčasnosti používaných bezpečných šifier. V rámci základných CSP, zo symetrických šifier podporuje štandardizované šifry DES, 3DES, RC2, RC4 a AES. Z týchto šifier je považovaná za bezpečnú, podľa Qualys SSL Labs, len šifra AES [17]. V základnej so systémom distribuovanej verzii CryptoAPI je algoritmus AES podporovaný len jedeným typom CSP, *Microsoft AES Cryptographic Provider*, označovaný v rámci CryptoAPI typom `PROV_RSA_AES` [18]. Jedinou asymetrickou šifrou v CryptoAPI, ktorou je možné priamo šifrovať užívateľom zvolené bloky dát je šifra RSA. Ostatnými asymetrickými algoritmami je možné len exportovať kľúče z vnútorného úložiska CSP.

Obsah kľúčov však nie je úplne neprístupný pre program využívajúci CryptoAPI. Kľúče je možné exportovať vo formátoch špecifických pre daný používaný CSP. Základné CSP obsiahnuté v CryptoAPI podporujú nasledujúce formáty na exportovanie a importovanie kľúčov, tzv. *blobs* [19]:

- PUBLICKEYBLOB,
- PRIVATEKEYBLOB,
- SIMPLEKEYBLOB,
- SYMMETRICWRAPKEYBLOB,
- PLAINTEXTKEYBLOB.

Štruktúra týchto formátov je detailnejšie popísaná v časti 3.2.3.

V CryptoAPI existujú 2 základné typy objektov, s ktorými je možné nepriamo manipulovať pomocou abstraktnej handle. Prvým typom objektov sú objekty hešov, reprezentované handlami typu `HCRYPTHASH`. Handle k objektu hešu je možné získať len dvomi funkciami CryptoAPI. Týmito funkciami sú [20]:

- `CryptCreateHash` – táto funkcia v CSP vytvorí prázdny objekt hešu, s algoritmom, ktorý je špecifikovaný v parametroch tejto funkcie,
- `CryptDuplicateHash` – táto funkcia požaduje parameter špecifikujúci už existujúci objekt hešu podľa jeho handle, následne je tento hash duplikovaný, vrátane jeho vnútorného stavu.

Po získaní handle k objektu hešu je možné daný heš používať pomocou funkcií `CryptHashData`, `CryptGetHashParam` a `CryptSetHashParam`. Taktiež je možné bezpečne zničiť daný vnútorný objekt hešu v CSP podľa jeho handle pomocou funkcie `CryptDestroyHash`.

Druhým typom objektov, s ktorými je možné vykonávať operácie ponúkané knižnicou CryptoAPI, je objekt kryptografického kľúča, s ktorým je manipulované v CSP pomocou handle typu `HCRYPTKEY`. Handle ku CryptoAPI kľúču je možné získať volaním následovných funkcií:

- `CryptGenKey` – táto funkcia v CSP náhodne vygeneruje kľúč, podľa špecifikovaného ID algoritmu. Podľa ID algoritmu určí typ algoritmu a veľkosť kľúča a vyplní jeho vnútorné hodnoty bezpečne náhodne vygenerovanými dátami. Výnimkou je algoritmus RSA, ktorý špecifikuje veľkosť kľúča v jednom z parametrov – flags, nastavením bitu v horných dvoch bajtoch tohto parametru.
- `CryptDuplicateKey` – táto funkcia zduplikuje už existujúci objekt kľúča, podľa jeho hodnoty handle, pritom skopíruje celý vnútorný stav kľúča, z ktorého je vytvorený náš nový kľúč.
- `CryptDeriveKey` – táto funkcia slúži na odvodenie kľúča z užívateľských dát. Odvodzovať kľúče je však nutné na základe objektu hešu, špecifikovaného ako parameter tejto funkcie pomocou jeho handle. Výhodou tohto spôsobu je, že na základe vstupných dát, ktoré boli vložené do objektu hešu, vieme presne odvodiť kľúčový materiál, pomocou tento kľúč následne šifruje dáta.
- `CryptImportKey` – je funkcia, ktorá vytvorí v CSP objekt kľúča na základe dát, ktorých formát určujé daný typ podporovaného formátu exportovaného či importovaného kľúča – *blobu*.
- `CryptImportPublicKeyInfo` – pomocou tejto funkcie je možné importovať kľúč formátovaný ako objekt `RSAPublicKey` podľa PKCS#1. Tento formát je popísaný nižšie v sekcii 2.3.
- `CryptGetUserKey` – touto funkciou je možné vytvoriť objekt kľúča v CSP jeho vyvolaním z kontajneru *key container*. Je tak možné získať handle ku kľúču či už uloženého ako `AT_SIGNATURE`, ale aj `AT_KEYEXCHANGE`.

Po vytvorení objektu kľúča a obdržaní jeho handle, je s ním možné vykonávať rôzne operácie. Dôležitými sú operácie na šifrovanie a dešifrovanie dát, `CryptEncrypt` a `CryptDecrypt`. Taktiež je možné nastavovať parametre existujúcich kľúčov pomocou funkcie `CryptSetKeyParam`. Pomocou tejto funkcie je užívateľ schopný na-

staviť napríklad mód operácie blokovej symetrickej šifry, alebo *inicializačný vektor* šifry. Kľúče môže tiež užívateľ exportovať s využitím funkcie `CryptExportKey`, ktorá umožňuje exportovanie vo formátoch *blob* zmieňovaných už vyššie. Poslednou z užitočných funkcií knižnice CryptoAPI s objektami kľúčov je funkcia na exportovanie kľúčov RSA `CryptExportPublicKeyInfo`, ktorá slúži na exportovanie verejného kľúča šifry RSA vo formáte `RSAPublicKey` definovaným v štandarde PKCS#1. Tento formát je popísaný ďalej v časti 2.3. Objekt kľúča je možné zničiť na základe jeho handle zavolaním funkcie `CryptDestroyKey`.

2.3 Asymetrická šifra RSA v CryptoAPI

Šifra RSA je asymetricé kryptografické primitívum, definované v *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications* [21] (PKCS#1 – RFC 3447). Využíva dva typy kľúčov: verejný kľúč a privátny kľúč, ktoré sú spolu nazvané ako kľúčový pár. Verejný kľúč RSA pozostáva z dvoch častí:

- n – nazývaný ako *modulus*, kladné celé číslo,
- e – verejný *exponent*, kladné celé číslo.

Jeden z podporovaných spôsobov importovania a exportovania RSA kľúčov v knižnici CryptoAPI je reprezentácia verejného kľúča podľa PKCS#1. Formát, v ktorom bývajú verejné kľúče RSA exportované, je v PKCS#1 definovaný pomocou značenia ASN.1, ktoré je špecifikované v štandarde ITU-T X.680 [22]. Verejný kľúč RSA je definovaný ako objekt typu `RSAPublicKey`.

Výpis 2.1: Definícia objektu RSAPublicKey v ASN.1 z PKCS#1 [21]

```
RSAPublicKey ::= SEQUENCE {
  modulus           INTEGER,  -- n
  publicExponent    INTEGER   -- e
}
```

Táto sekvencia je následne kódovaná do binárnej sekvencie oktetov podľa pravidiel kódovania DER (Distinguished Encoding Rules) definovaných v štandarde *ITU-T X.690* [23].

3 Návrh metód detekcie použitej kryptografie v binárnom kóde

Hlavnou motiváciou tejto práce je zjednodušenie a zrýchlenie analýzy programu využívajúceho prostriedky kryptografie, pretože doteraz bolo potrebné túto analýzu vykonávať manuálne pomocou iných prostriedkov dynamickej či statickej analýzy. V predchádzajúcich častiach bola popísaná v súčasnosti stále využívaná knižnica CryptoAPI, distribuovaná so systémami Microsoft Windows, a nástroj na automatickú dynamickú analýzu programov, sandbox Cuckoo. Podľa v tejto práci získaných údajov, približne 7,11 % zo všetkých súborov analyzovaných v tomto sandboxe, využíva počas svojho behu nejakým nešpecifikovaným spôsobom knižnicu CryptoAPI. Existuje tak príležitosť rozšíriť analýzu týchto vzoriek programov, využívajúcich prostriedky CryptoAPI a prechádzajúcich automatickou analýzou v sandboxe Cuckoo.

Prínos výstupu tejto práce môže byť v zjednodušení interpretácie volaní funkcií z knižnice CryptoAPI a následné poprepájanie súvisiacich volaní do spoločných celkov vo výstupe z analýzy. Predíde sa tak nutnosti vyhľadávania jednotlivých volaní CryptoAPI v hrubom výstupe analýzy (*Behavioral log* v časti 1.1.3), ktoré často samé o sebe bez kontextu ostatných majú nízku výpovednú hodnotu. Pospájaním spolu súvisiacich volaní sa však výpovedná hodnota, získateľná z analýzy, zvyšuje a je tak možné si z nej vytvoriť lepšiu predstavu o využití prostriedkov knižnice a posúdiť prípadné zneužitie prostriedkov kryptografie na spôsobenie škody obeti. Je tak možné spozorovať viacero rôznych chovaní, ktoré môžu vypovedať o úmysloch neznámeho programu. Medzi chovania, ktoré je možné takýmto spôsobom odhaliť patria napríklad šifrovanie užívateľských dát, šifrovanie dát verejným kľúčom, ktorý nebol lokálne vytvorený, alebo šifrovanie exportovaných kľúčov inými kľúčmi, popísaným neskôr v tejto kapitole.

Na základe toho, že funkcie CryptoAPI sú poskytované výhrade cez rozhrania poskytované knižnicami DLL (viď kapitola 2.2), je možné využiť využiť možnosti poskytované sandboxom Cuckoo na hookovanie (viď kapitola 1.1.3) konkrétnych volaní exportovaného rozhrania knižníc DLL poskytujúcich rozhranie CryptoAPI. Pomocou funkcionality Cuckoo na zaznamenávanie volaní a ich parametrov je následne možné vo vlastnom module *Signature* analyzovať pozberané dáta z vybraných sledovaných volaní funkcií knižnice CryptoAPI.

CryptoAPI bude následne predstavovať čiernu skrinku (black box), ktorá sa vo vnútri chová ako stavový automat, ku ktorého vstupy a výstupy sú zachytávané. Na základe týchto zozberaných vstupov a výstupov je možné zhruba interpretovať, čo sa odohrávalo vo vnútri knižnice. Keďže väčšina dát je ukrytá vo vnútri čiernej skrinky CryptoAPI, priamy prístup k dátam, ako napríklad hodnota kľúča či inicializačného

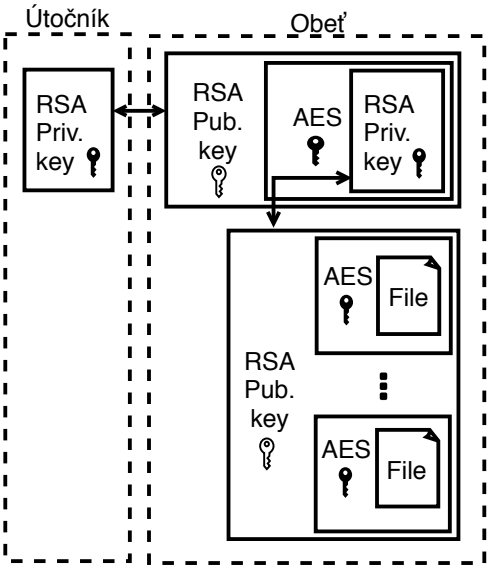
vektora, s ktorými sa operovalo, nie je možný, pokiaľ neboli zachytené a odhalené pri importovaní, či exportovaní. Zľahčujúca je však vlastnosť CryptoAPI, kedy je manipulovanie s objektami CryptoAPI možné len po ich vytvorení príslušným volaním CryptoAPI na vytváranie objektov. Ako bolo už v kapitole 2.2 zmienené, tieto volania sú nám známe a je tak možné ich sledovať. Následne je možné, podľa použitého *handlu* `HCRYPTKEY` či `HCRYPTHASH`, vo volaní operácie s objektom tento objekt identifikovať a zistiť jeho prepojenosť s volaním, ktorým bol vytvorený, a inými predchádzajúcimi operáciami. Takto je možné dostatočne popísať, aké objekty boli pri daných operáciach použité, z akého volania vznikli, a či sú známe dáta, pomocou ktorých boli tieto objekty vytvorené alebo odvodené.

Počas práce vytvorený analyzátor knižnice CryptoAPI by tak mal byť schopný reagovať na všetky volania CryptoAPI, určené na manipuláciu s elementárnymi objektami, a mal by zberať informácie potrebné na jednoznačnú identifikáciu a popis spôsobu využitia kryptografických prostriedkov. Ako základné rozhranie so systémom Cuckoo bude autorom vytvorená signatúra v module Signature sandboxu Cuckoo, ktorá bude mať medzi zaregistrovanými funkciami práve funkcie CryptoAPI. Všetky volania následne posunie objektu triedy `CryptoAnalyser`, ktorý bude zastrešovať reprezentáciu aktuálnych stavov objektov CryptoAPI a následné prezentovanie ich aktivity po spracovaní všetkých funkcií.

Základnou požiadavkou je schopnosť automaticky spracovať aktivitu využitia kryptografických prostriedkov CryptoAPI a jej následnú prehľadnú a intuitívnu prezentáciu, pomocou ktorej bude urýchlená analýza programu, ktorý využíva tieto prostriedky. Analýzu je ešte možné vylepšiť o analýzu a schopnosť detekcie manipulácie s kľúčmi. Vo viacerých prípadoch, napríklad pri ransomware Spora, Rapid, či Alcatraz, ktoré boli využité pri vývoji v tejto práci vytvoreného analyzátora, sa používa viacero kľúčov, kedy pomocou jedného kľúča sú šifrované exportované iné kľúče. Hlavným dôvodom je následná možnosť distribúcie a výmeny len jedného kľúča, ktorým bude obeť schopná dešifrovať svoje súbory, ktoré tak môžu byť šifrované samostatnými unikátnymi kľúčmi. Takto útočník získa istotu, že po zašifrovaní súboru obete, obeť nebude schopná získať kľúč na jeho spätné dešifrovanie, pretože už nebude na nič iné potrebný a jeho pravá hodnota bude znehodnotená.

Príklad tohto obaľovania a hierarchie použitých kľúčov je ukázaný na obrázku 3. Jednotlivé súbory sú šifrované samostatne unikátnymi symetrickými kľúčmi šifry AES-256. Následne tieto kľúče, ktorými boli šifrované dáta súborov, sú šifrované lokálne vygenerovaným RSA párom kľúčov, ktorého súkromný kľúč je zašifrovaný jedným „hlavným“ kľúčom šifry AES-256. Týmto je zabezpečené, že všetky už zašifrované súbory nebude možné dešifrovať bez znalosti súkromného kľúča, pretože ho je v dobe šifrovania už nemožné získať. Tento „hlavný“ kľúč je zašifrovaný druhým RSA párom kľúčov, ktorého súkromný kľúč je držaný autorom ransomwaru, a teda

je tajomstvom, pomocou ktorého je možné postupne získať všetky kľúče vedúce až ku kľúčom, ktoré boli použité na zašifrovanie dát obete. Verejný kľúč z tohto páru je pevne uložený v zdrojovom kóde programu a program je s ním distribuovaný.



Obr. 3.1: Príklad schémy „obaľovania“ kľúčov ďalšími a vytváranie tak reťazca na sebe závislých kľúčov u ransomwaru Spora

Jednoduchosť a intuitívnosť výstupu analyzátora bude zabezpečená tým, že analyzátor bude zobrazovať obsahy bufferov[1], ktoré boli šifrované či hešované pomocou objektov CryptoAPI. Výstup u šifrovaných bufferov je možné rozšíriť o informáciu, či bol daný buffer rozpoznaný ako v minulosti už vytvorený a exportovaný kľúč. Pomocou takto identifikovanej hierarchie kľúčov bude možné identifikovať, s akým „hlavným kľúčom“ bol pravdepodobne distribuovaný analyzovaný program. Taktiež dôležitou časťou výstupu je rozpoznávanie a prezentovanie pôvodu kľúča. Je takto možné zistiť, či je možné prelomiť použité šifrovanie a vytvoriť naň dekryptor, a či nie je použitá schéma šifrovania napadnuteľná iným spôsobom. Keďže začiatok každého buffera bude overovaný na rozpoznanie existujúceho kľúča, je možné analyzátor rozšíriť aj o schopnosť rozpoznať, či ide o užívateľský súbor pomocou signatúr súborov, špecifikovaných ako regulárne výrazy úvodných bajtov daného typu súboru. Táto schopnosť tak bude upozorňovať na pravdepodobnú manipuláciu s užívateľskými dátami formátu ako sú ZIP, DOC, JPEG a iné.

V prípade analýzy vzorky, ako je ransomware, môže byť vytvorené až príliš veľké množstvo kľúčov a je dôležité, pre zachovanie jednoduchosti a výpovednej hodnoty

[1]Časti pamäte, s ktorými sa operuje, zvyčajne identifikované ako adresa začiatku a dĺžka vyhradenej pamäte.

výstupu, ich prezentáciu obmedziť tak, aby nebola analýza príliš veľká a tým ťažko spracovateľná pre človeka. Avšak nemôže byť obmedzená až tak, že by mohla spôsobiť zahodenie dôležitých informácií. Preto pri prezentácii získaných dát, budú rozlišované operácie s kľúčmi asymetrických šifier a symetrických šifier. Kľúče v týchto kategóriách budú zobrazované podľa času ich vytvorenia, takže kľúče, ktoré by bolo možné označiť ako „hlavné“ (tj. boli nimi šifrované iné dôležité kľúče), sa s najväčšou pravdepodobnosťou objavia na vrchu zoznamu manipulovaných kľúčov. Počet zobrazených kľúčov vo výstupe bude nastaviteľný, aby jeho úprava bola čo najmenej invazívna do štruktúry analyzátora.

3.1 Príprava Cuckoo monitor knižnice

Na schopnosť úspešne interpretovať aktivity knižnice CryptoAPI bude potrebné hookovať všetky jej dôležité funkcie. V časti 2.2 sú už definované vymenované funkcie predstavujúce dôležité operácie knižnice CryptoAPI. Je potrebné vytvoriť stabilné rozhranie, pomocou ktorého bude možné identifikovať parametre jednotlivých hookovaných funkcií a podľa ich názvu s nimi pracovať v rámci v tejto práci vytvoreného analyzátora. Preto v rámci príprav prostredia pre analyzátor aktivity CryptoAPI bola knižnica Cuckoo monitor rozšírená o hooky funkcií, ktoré doteraz nesledovala, a v niektorých hookoch boli pridané nové zberané údaje na umožnenie interpretácie zberaných dát.

Funkcie CryptoAPI sú poskytované dvomi knižnicami DLL. Sú nimi `advapi32.dll` a `crypt32.dll`. CryptoAPI funkcie z knižnice `advapi32.dll` môžu niekedy byť nahrané až oneskorene počas behu programu, preto je nutné ich hookovať aj v jej vnútornej knižnici `cryptsp.dll`. Zoznam všetkých hookovaných funkcií, v podobe, v ktorej je s nimi v rámci tejto práce narábané, a ich zberané parametre spolu s názvami, pod ktorými sú následne dostupné, sú podrobne rozpísané na priloženom CD (príloha C) vo formáte popísanom v prílohe B. V tejto práci vytvorený analyzátor bude silno zviazaný s touto definíciou zberaných parametrov volaní. Vstupné funkcie analyzátora budú mať k dispozícii len parametre vyslovene zaznamenávané v hookoch, v type, akom sú deklarovné a pod menom, ktoré je mu v zaznamenávajúcej funkcii priradené.

3.2 Spracovanie vybraných volaní CryptoAPI

Keďže sa operácie vykonávané s objektami kľúčov a hešov dajú oddeliť, bude spracovanie ich funkcií opísané oddelene. Pre každú funkciu z oboch kategórií budú opísané kroky potrebné na zabezpečenie presnej analýzy, ktoré budú následne uplatnené pri

implementácii automatického analyzátora ako signatúry v Cuckoo sandboxe. Pri definícii krokov, ktoré je potrebné vykonať na vytvorenie úspešnej interpretácie aktivity s knižnicou CryptoAPI, je vychádzané predovšetkým z postupu, ktorý by bol použitý pri manuálnej analýze programu operujúceho s knižnicou CryptoAPI.

3.2.1 Emulácia funkcionality CSP

Ako bolo už v kapitole 2.2 opísané, prvým krokom pred manipuláciou s objektami CryptoAPI je vždy získanie kontextu CSP. To je možné docieliť zavolaním buď funkcie `CryptAcquireContextA`, či `CryptAcquireContextW`. Tak ako aj u iných funkcií knižníc Microsoft, jediným rozdielom medzi týmito funkciami je, že funkcia končiaca na -A prijíma reťazce typu pole znakov `CHAR` a funkcia končiaca na -W akceptuje reťazce typu pole znakov `WCHAR`. Obe funkcie majú ostatné parametre identické. Preto ich obe bude v analyzátore spracovávať totožná funkcia. Funkcie vracajú 32-bitový handle typu `HCRYPTPROV`, ktorý je potom uplatňovaný pri tvorbe objektov oboch typov, hešov i kľúčov. Kontext CSP je možné uvoľniť funkciou `CryptReleaseContext` pomocou predtým získanej handle.

Pre analyzátor je emulácia funkcionality CSP dôležitá, aby bolo možné správne interpretovať volania funkcií operujúcich s kľúčmi aktuálne uloženými v kontajneri kľúčov (*key container*, opísaný v kapitole 2.2). Ide o funkcie `CryptGetUserKey`, ktorou je možné získať handle ku jednému z aktuálne uložených kľúčov v kontajneri kľúčov, potom funkciu `CryptExportPublicKeyInfo`, ktorou je možné exportovať jeden z aktuálne v kontajneri uložených kľúčov vo formáte podľa štandardu PKCS#1. Tento spôsob je popísaný podrobnejšie v časti 2.2. Bez emulácie aktuálne uložených kľúčov v práve otvorených kontextoch s ich kontajnermi by mohla nastať situácia, kedy by sa v zázname volaní mohli objaviť operácie s kľúčom, ktorého pôvod je neznámy alebo by nebolo možné priradiť exportovaný kľúč k jeho reprezentácii v analyzátore.

Emulácia funkcionality CSP prebieha tým, že súčasťou analyzátora bude centrálny objekt reprezentujúci kontajnery kľúčov, v ktorom budú spravované všetky rozpoznané a zaznamenané použité kontajnery kľúčov. Pri vytvorení kontextu je podľa parametru názvu kontajneru kľúčov priradený kontextu CSP jeho kontajner a všetky, v daný moment posledné vytvorené, asymetrické kľúče sú uložené v ich patričnom odkaze, tj. `AT_SIGNATURE` alebo `AT_KEYEXCHANGE`. Ak by program pri vytvorení ďalšieho kontextu CSP požadoval rovnaký kontajner, bolo by zabezpečené to, že bude mať prístupné práve v kontajneri uložené kľúče a bude možné sledovať a správne interpretovať operácie s nimi.

3.2.2 Interpretácia operácií s objektami hešov CryptoAPI

Funkcie používajúce objekty hešov sa dajú rozdeliť na 3 skupiny: funkcie vytvárajúce objekty hešov, funkcie operujúce s hešmi a funkcia ničiaca existujúci objekt hešu.

Funkcie vytvárajúce heše sú funkcie `CryptCreateHash` a `CryptDuplicateHash`. Ich bližší popis je v časti 2.2. Pri týchto funkciách bude vo vnútri analyzátora vytvorený objekt emulujúci objekt hešu vo vnútri CryptoAPI CSP. Pri duplikácii objektu volaním funkcie `CryptDuplicateHash` stačí zabezpečiť vytvorenie kópie už existujúceho objektu hešu s jeho celým vnútorným stavom.

Pri funkciách operujúcich s existujúcimi objektami hešov je potrebné v navrhovanom analyzátore zabezpečiť, že sú dané operácie vykonávané so správnymi objektami hešov. Pri volaní `CryptHashData` je obsah bufferu pridávaný do simulovaného vnútorného objektu hešu, pretože sa buffery z viacerých volaní spájajú, až do takzvanej *finalizácie* objektu hešu [24]. Tá nastane zavolaním funkcie `CryptGetHashParam` s parametrom `HP_HASHVAL`. Vtedy teda treba zabezpečiť, že objekt hešu sa *finalizuje* a nebude tak doň možné už neskôr pridávať vstupné dáta pomocou volania `CryptHashData`.

3.2.3 Interpretácia operácií s objektami kľúčov CryptoAPI

U kľúčov existuje šesť spôsobov na vytvorenie objektu kľúča v CSP CryptoAPI. Najjednoduchším spôsobom na interpretáciu je volanie `CryptGenKey`, ktoré vo vnútri CSP vytvorí nový kľúč s náhodne vygenerovanými dátami na šifrovanie. Vtedy stačí iba poznamenať a výstupe upozorniť, že daný nový objekt kľúča je náhodne vygenerovaný. Pokiaľ takto vytvorený kľúč nie je niekedy exportovaný, nie je možné získať jeho hodnotu. Avšak informácia, že daný kľúč bol generovaný lokálne a rozpoznanie algoritmu sú stále užitočné. Presná číselná hodnota kľúča nie je potrebná z hľadiska analýzy úmyslu programu, pretože je v každom prípade špecifická pre daný konkrétny beh programu a stačí vedieť, že je spoľahlivo náhodná pri každom behu. Výnimkou pri analýze volania funkcie `CryptGenKey` je generovanie páru kľúčov RSA (u ostatných je dĺžka určená na základe identifikátoru algoritmu), kedy je v veľkosť kľúču špecifikovaná ako dodatočný parameter funkcie.

Ďalším spôsobom na vytvorenie objektu kľúča je volanie `CryptDuplicateKey`, ktoré zduplikuje objekt kľúča aj s celým jeho vnútorným stavom, ktorý predstavujú aj vektory používané pri prepájaní blokov pri móde šifry ako CBC a podobne. Interpretácia tejto operácie je z pohľadu analyzátora triviálna, taktiež je možné poznamenať pôvod (tj. objekt použitý ako predloha na duplikáciu) tohto kľúča.

Vytvorenie objektu kľúča pomocou volania `CryptGetUserKey` je spracovávaný na základe predaného parametru kontextu CSP. Na základe jeho hodnoty je nájdený vnútorne v analyzátore simulovaný kontext a získaný jeho kontajner kľúčov.

Následne je možné z tohto kontajneru získať daný objekt kľúča, ktorý bol týmto volaním získaný, a vytvoriť jeho kópiu.

Posledným z volaní triviálnych na spracovanie je `CryptDeriveKey`. Pri tomto volaní musí analyzátor identifikovať správny objekt hešu, z ktorého je kľúč odvodený a postarať sa o finalizovanie tohto objektu hešu [25]. Na základe prepojenia s objektom hešu je možné presne zistiť výsledok hešu, z ktorého bol šifrovací kľúč odvodený podľa známeho algoritmu opísaného v dokumentácii [25]. Na základe toho, síce nie je zachytený výsledný kľúč, je možné považovať šifrovací kľúč za známy, pretože z hešu konštantných dát je aj odvodený kľúč konštantný.

Volania na importovanie kľúčov z binárnych dát majú ako zaznamenávaný parameter len binárne zakódovanú informáciu o kľúči, preto na identifikáciu algoritmu či šifrovacieho kľúča je nutné tieto zaznamenané binárne dáta spracovať podľa pravidiel nižšie opísaných. Importovanie kľúčov zabezpečuje funkcia `CryptImportKey` a čiastočne `CryptImportPublicKeyInfo`, ktoré akceptujú rôzne binárne reprezentácie kľúčov.

Spracovanie importovaných kľúčov

Ako bolo už v kapitole 2.2 opísané, funkcia `CryptImportKey` akceptuje ako parametre viacero formátov reprezentácie kľúča určeného na import. Všetky z týchto formátov majú spoločný začiatok. Začiatok každého `BLOBu` tvorí osembajtová (kedy pod bajt je myslený oktet bitov) štruktúra `PUBLICKEYSTRUC` [26].

Výpis 3.1: Deklarácia formátu spoločnej hlavičky `BLOBov`

```
typedef struct _PUBLICKEYSTRUC {
    BYTE   bType;
    BYTE   bVersion;
    WORD   reserved;
    ALG_ID aiKeyAlg;
} BLOBHEADER, PUBLICKEYSTRUC;
```

Podľa prvého bajtu je možné rozoznať typ `BLOBu`, navrhovaný analyzátor sa zaoberá nasledujúcimi typmi (ostatné, ktoré neposkytujú základné CSP, sú dostupné v prílohe A.3):

- 0x1 – `SIMPLEBLOB` – obsahuje kľúč symetrickej šifry zašifrovaný algoritmom asymetrickej šifry.
- 0x6 – `PUBLICKEYBLOB` – predstavuje verejný kľúč asymetrickej šifry v planej podobe.
- 0x7 – `PRIVATEKEYBLOB` – obsahuje celý kľúčový pár s hodnotami stanovenými v štandarde PKCS#1, avšak nie sú reprezentované v kódovaní DER.

- 0x8 – PLAINTEXTKEYBLOB – obsahuje kľúč symetrickej šifry v otvorenej forme.
- 0xb – SYMMETRICWRAPKEYBLOB – obsahuje kľúč symetrickej šifry šifrovaný iným symetrickým kľúčom.

Bajt bVersion má u v súčasnosti používaných CSP hodnotu vždy 2. Štvorbajtová hodnota čísla aiKeyAlg unikátne identifikuje algoritmus použitého kľúča. Po hlavičke PUBLICKEYSTRUC nasledujú dáta špecifické pre konkrétny blob.

Najjednoduchším z pohľadu obsahu je blob typu PLAINTEXTKEYBLOB, ktorý po hlavičke blobu obsahuje štvorbajtové číslo určujúce počet nasledujúcich bajtov dát kľúča. Pri importovaní kľúča vo formáte PLAINTEXTKEYBLOB je tak dostačujúce rozpoznať hlavičku a následne číslo určujúce dĺžku binárnych dát kľúča.

Kľúč importovaný ako SIMPLEBLOB je zašifrovaný verejným kľúčom RSA, na jeho dešifrovanie je potrebné predať v parametre handle k ďalšiemu objektu kľúča, ktorý musí predstavovať dešifrovací kľúč k použitému verejnému kľúču. Ak je teda importovanie kľúču v SIMPLEBLOB úspešné, je potrebné zaznamenať väzbu medzi importovaným kľúčom a kľúčom použitým na jeho dešifrovanie.

Kľúče vo formáte PUBLICKEYBLOB a PRIVATEKEYBLOB sú v tejto práci obmedzené na kľúče RSA, pretože CryptoAPI neponúka podobnú funkcionalitu s inými asymetrickými prostriedkami a jedinou zdokumentovanou podobou je podoba definovaná pre šifru RSA. Majú okrem hlavičky blobu spoločnú aj hlavičku RSA kľúča RSAPUBKEY, definovanú ako štruktúru vo výpise 3.2. Prvé štvorbajtové číslo zvané magic definuje, či ide o PUBLICKEYBLOB – vtedy je jeho hodnota 0x31415352 (reťazec „RSA1“ v ASCII), alebo PRIVATEKEYBLOB – vtedy je jeho hodnota 0x32415352 (reťazec „RSA2“ v ASCII). Ďalšie číslo v štruktúre, bitlen, udáva bitovú dĺžku kľúča. Posledný člen štruktúru, pubexp, predstavuje verejný exponent z verejného kľúča (v CryptoAPI je preferované prvočíslo 65537 ako tento exponent). Následne po tejto štruktúre nasleduje hodnota modulu RSA kľúča s dĺžkou bitlen/8 bajtov. Všetky tieto informácie je možné pri importovaní získať. Je dôležité ich v analyzátore uchovať a následne na konci prezentovať.

Výpis 3.2: Deklarácia spoločnej hlavičku u kľúčov vo formáte PUBLICKEYBLOB a PRIVATEKEYBLOB

```
typedef struct _RSAPUBKEY {
    DWORD magic;
    DWORD bitlen;
    DWORD pubexp;
} RSAPUBKEY;
```

Posledný spôsob na importovanie kľúča je funkcia CryptImportPublicKeyInfo, ktorá ako parameter očakáva objekt RSAPublicKey opísaný v časti 2.3. V danom prípade je treba spracovať binárne dáta vo formáte kódovania DER. Zo získaných

dát je možné získať veľkosť použitého kľúča a verejný exponent, ktoré je možné pridať do výsledku analýzy daného objektu kľúča CryptoAPI.

Formát `SYMMETRICWRAPKEYBLOB` využíva parameter volania `CryptImportKey` zvaný `hPubKey`, ktorý v danom prípade reprezentuje handle k symetrickému kľúču. Vtedy je potrebné len priradiť k dátam pôvodu kľúču informáciu o kľúči, ktorým bol obalený importovaný kľúč. Tento formát nie je podporovaný s kľúčmi šifry AES, či už ako obaľovanom kľúči alebo obaľujúcom kľúči.

Operácie s existujúcimi kľúčmi

Následne po získaní kľúča je potrebné zaznamenávať jeho operácie a prípadne pri nich meniť vnútorný stav. Najdôležitejšie funkcie sú `CryptEncrypt` a `CryptDecrypt`. U šifrovania je potrebné zberať vstupné dáta pred ich zašifrovaním, u dešifrovania výstupné dáta dešifrovania. U oboch z týchto funkcií je parameter `Final`, ktorým sa signalizuje posledný šifrovaný blok. Slúži na signalizáciu, že nepribudnú ďalšie dáta, a CryptoAPI na základe tohto parametru pri nenulovej hodnote pridáva výplň (padding) na koniec bloku. Ak sú šifrované či dešifrované dáta pomocou častí volaním funkcie viackrát, pre všetky časti dát je hodnota nula, až na posledný blok.

Preto pri prvom bloku, ktorý nie je označený ako posledný je zvyšované aj počítadlo využití kľúča. Všetky následné volania šifrovania či dešifrovania sú prepojené s týmto blokom a toto prepojenie preruší blok s nenulovou hodnotou `Final`.

Volaním `CryptSetKeyParam` je možné upraviť parametre kľúča. Dôležitými parametrami, ktoré je potrebné sledovať sú *inicializačný vektor* a *operačný mód* ak ide o blokovú šifru. Avšak z dôvodu nevhodného rozhrania, nie je možné bez volaní ďalších funkcií z vnútra tela hooku extrahovať hodnotu inicializačného vektoru, ak je nastavovaný na inú ako pôvodnú hodnotu. Je to preto, lebo funkcii je predaný len ukazovateľ na pamäť, kde je nový inicializačný vektor. Vtedy však nie je vo vnútri hooku možné zistiť, ako veľká časť pamäte je potrebná či využitá. Mohlo by sa tak stať, že by sa monitorovacia knižnica za účelom zistenia inicializačného vektoru pokúsila čítať neprístupnú pamäť a tým spôsobiť výnimku, ktorá by analyzovaný program ukončila a tak by mohla byť analýza predčasne ukončená a znehodnotená. Na základe tohto rizika teda nie je implementované rozpoznávanie nových inicializačných vektorov. Preto pri tomto volaní je len detegovaný a rozpoznávaný parameter zmeny operačného módu šifry, ktorý je možné rozpoznať konštantou `KP_MODE` v parametre `dwParam` funkcie. Vtedy je v tele hooku zaznamenaný nový použitý operačný mód identifikovaný jednou z konštánt z výpisu A.2.

Kľúče je možné exportovať pomocou volaní `CryptExportKey` a `CryptExportPublicKeyInfo`, ktoré exportujú kľúče vo formátoch ich importovacích alternatív `CryptImportKey` a `CryptImportPublicKeyInfo`. Pri týchto ope-

ráciach nie je analyzovaný obsah výstupných binárnych dát, pretože by vlastnosti kľúča mali byť zjavné na základe detekovaného typu `BLOBu` a popisu pôvodu kľúča. Stačí teda rozoznať o ktorý formát výstupu ide, čo sa dá na základe dodatočného parametru u volania `CryptExportKey` označujúceho požadovaný formát, nazvaného ako `dwBlobType`.

Exportované kľúče budú zaznamenávané nielen vrámci jednotlivých vnútorných reprezentácií kľúčov CryptoAPI, ale zároveň aj v spoločnom kontajnery slúžiacom na porovnanie už s minulými exportami pri volaniach `CryptEncrypt`, či importovaní na identifikáciu opakovaného importovania či šifrovania v minulosti počas behu exportovaného kľúča.

4 Implementácia signatúry v sandboxe Cuckoo

Implementáciu v tejto práci navrhnutého analyzátora reprezentovaného signatúrou v sandboxe Cuckoo je možné rozdeliť na dva samostatné celky. Prvá časť sa zaoberá analýzou a interpretovaním dostupných informácií z API volaní a budovaním vhodnej štruktúry, ktorá bude reprezentovať zachytenú aktivitu s prostriedkami knižnice CryptoAPI. Následne, druhá časť sa zaoberá prezentáciou získaných informácií vo výsledku analýzy Cuckoo a integráciou do webového prostredia Cuckoo.

4.1 Architektúra analyzujúcej časti

Navrhovaný analyzátor je implementovaný ako samostatná signatúra v module *Signature* sandboxu Cuckoo. Na oddelenie analyzátora od architektúry sandboxu Cuckoo, je reprezentovaný ako objekt triedy `CryptoAnalyser`, ktorý poskytuje rozhrania potrebné pre signatúru. Jeho hlavným rozhraním je funkcia `on_call`, ktorá je volaná z funkcie `on_call` signatúry Cuckoo. Funkcia `on_call` signatúry posúva ďalej svoje parametre analyzátoru, s tým, že parametre API volania, s ktorým bola práve zavolaná upraví na ľahšie použiteľný formát nevyžadujúci hľadanie lineárnym priechodom a pridá ich ako ďalšie parametre `arguments` a `raw_arguments` pre funkciu `on_call` objektu `CryptoAnalyser`.

Objekt príjme toto volanie a na základe predaného parametru `process`, ktorý reprezentuje popis procesu, v ktorom dané spracovávané volanie bolo volané vo forme Python typu `dictionary`, rozhodne ktorému vnútornému objektu reprezentujúceho proces má byť dané volanie predané. Objekt `CryptoAnalyser` drží kontajner `processes`, v ktorom je možné vyhľadávať jednotlivé vnútorné reprezentácie procesu podľa ich ID. Ak je práve spracovávané API volanie prvé z daného procesu, je proces pridaný do kontajneru `processes`. Vnútorná reprezentácia procesov je použitá na separáciu objektov CryptoAPI, pretože by mohlo dochádzať ku kolíziám *handlov* medzi procesmi, čo nie je v CryptoAPI možné, pretože handle sú platné len lokálne pre ich konkrétne procesy. Procesu je toto volanie predané ďalej volaním jeho metódy `on_call`.

Každý vnútorný objekt reprezentujúci proces obsahuje dva objekty, po jednom z tried: `HashManager`, objekt spravujúci objekty hešov, a `KeyManager`, objekt spravujúci objekty CryptoAPI kľúčov. Pri spracovaní API volania, proces rozhodne, o ktorý typ volania ide na základe definovaných množín názvov API, ktoré spadajú pod hešovacie objekty resp. objekty kľúčov CryptoAPI. Tieto definície sú vo výpise A.1. Následne zavolá podľa názvu aktuálneho volania príslušnú funkciu objektu

`HashManager` resp. `KeyManager` zodpovednú za spracovanie volania podľa postupov popísaných v časti 3.2. Tieto funkcie spracovávajúce konkrétne CryptoAPI volania tvoria metódy nazvané ako `on_<Názov_volania>`, ako napríklad `on_CryptGenKey`, definované vrámci príslušného spravujúceho objektu.

Objekty zodpovedné za správu objektov hešov a kľúčov tieto kľúče ukladajú pomocou dvoch typov kontajnerov. Objekty sú vkladané postupne do zoznamu zachovávajúceho poradie, v ktorom boli tieto objekty pridané. U kľúčov sú použité dva zoznamy, samostatne pre kľúče symetrických a asymetrických algoritmov. Obsah tohto kontajneru je pre analýzu nepodstatný, svoju úlohu plní pri prezentácii výsledkov, avšak musí byť napĺňaný počas analýzy volaní. Druhým typom kontajneru použitého v objektoch spravujúcich objekty kľúčov resp. hešov hešovaný kontajner, v tomto prípade Python kontajner `dictionary`, za účelom prístupu k objektom kľúčov či hešov podľa hodnoty ich handle, ktorá v analyzujúcej signatúre predstavuje reťazec s hodnotou handle v šestnástkovej sústave, napríklad `'0x00001234'`. Je takto zaručená vysoká rýchlosť prístupu k objektom podľa handle aj v prípade veľkého množstva vytvorených objektov.

CryptoAPI môže po zničení objektu v CSP znovu prideliť novému objektu hodnotu jeho handle, takže pre konkrétnu hodnotu handlu je zaznamenávaný lineárny zoznam kľúčov, ktorým bol tento handle priradený. Vždy je handle daného objektu unikátny, a preto pri analýze je vždy pristupované len k poslednému objektu s danou hodnotou handle. Preto aj odkazy na objekty vo výstupe signatúry sú reprezentované ako handle-index, kedy index predstavuje poradie v čase medzi objektami s rovnakou hodnotou handlu.

Správca objektov kľúčov taktiež obsahuje hešovaný kontajner na rýchle vyhľadávanie v zozname predošlých exportov kľúčov. Je tak možné pri každom šifrovanom bufferi a importovanom kľúči rýchlo vyhľadať, či neexistuje zhoda s predošlým exportom. Táto kontrola urýchľuje analýzu v prípade už skôr opísaného obaľovania kľúčov, pretože na základe tejto detekcie je možné získať predstavu o hierarchii kľúčov.

Vnútorné stavy kľúčov a hešov sú reprezentované pomocou fundamentálnych typov jazyku Python, hlavne `int` a `string`, a pomocou datatypov `enumeration` reprezentujúcich numerické konštanty. Objekty reprezentujúce objekty kľúčov a hešov poskytujú ich spravujúcim objektom rozhrania vychádzajúce z účinkov volaní popísaných v časti 3.2. Spracovanie binárnych dát je realizované pomocou knižnice `struct` a spracovanie DER kódovaných verejných RSA kľúčov knižnicou `asn1parse` dostupnou vo verejnom repozitári knižníc Python PyPi. Toto spracovanie prebieha vrámci metód spravujúcich objektov a objekty reprezentujúce dané objekty už dáta len konzumujú.

4.2 Prezentácia výsledkov signatúry

Spôsob prezentácie výsledkov signatúry je závislý na šablónach frameworku Django, ktorý sprostredkováva webové rozhranie Cuckoo. Tieto šablóny očakávajú prítomnosť kontajnera `data` alebo `new_data` v objekte detekovanej signatúry. Ako bolo už časti 1.1.3 ukázané, zobrazovanie informácií kontajneru `data` je nedostatočné na prezentáciu nazbieraných dát vytvoreného analyzátora, a preto je na zobrazovanie výsledkov využitý spôsob využívajúci kontajner `new_data`. Tento kontajner musí splňovať formát, pre ktorý je definovaná šablóna na jeho vykresľovanie. Keďže k tomuto spôsobu chýba dokumentácia, bolo potrebné túto požadovanú štruktúru získať analýzou šablóny použitej na vykresľovanie obsahu kontajneru `new_data` signatúry.

Výpis 4.1: Definícia štruktúry kontajneru obsahujúceho prezentované výstupné dáta pomocou pseudo-gramatiky

```
new_data := list[match]

match := dict {    # podtabuľka s vlastnou hlavičkou
  'process': process_summary,
  'signs': process_signs
}

process_summary := dict { # hlavička podtabuľky
  'process_id': int,
  'process_name': str
}

process_signs := list[sign] # zoznam riadkov podtabuľky

sign := dict {          # riadok v tabulke na výstupe
  'type': str,          # ľavá bunka
  'value': sign_data    # pravá bunka
}

# páry kľúčov a hodnôt v danom riadku
sign_data := str or dict[str,str] or list[pair[str,str]]
```

Šablóna očakáva `new_data` ako lineárny zoznam, ktorý pozostáva z objektov `dictionary` reprezentujúcich jednu samostatnú tabuľku vo výstupe, nižšie vo výpise označená ako objekt `match`. Obsahom tohto objektu je `dictionary` popisujúci ID procesu a jeho meno, ktoré budú zobrazené v hlavičke tabuľky. Týmto spôsobom je možné oddelene reprezentovať jednotlivé procesy vo výstupe analýzy. Druhým objektom objektu `match` je lineárny zoznam pozostávajúci z objektov `dictionary` reprezentujúcich jednotlivé riadky tabuľky, ktoré sú nazvané ako `sign`. Objekt `sign`

reprezentuje jeden riadok vykresľovanej tabuľky. Pozostáva z dvoch hodnôt, `type`, ktorý je vykreslený v ľavej bunke riadku, a `value`, ktorá predstavuje obsah pravej bunky riadku. Obsah `value` môže byť jeden reťazec, alebo `dictionary` predstavujúci hodnoty kľúč–hodnota, vypísané ako „**kľúč:** hodnota“. Tento spôsob mal nevýhodu v tom, že v jazyku Python 2 sú kľúče v objekte `dictionary` zoradené pseudonáhodne podľa hešu kľúčovej hodnoty. Preto bola šablóna určujúca formát HTML tabuľky vo výstupe signatúry upravená tak, aby bola schopná zobrazovať ako položku `value` aj zoznam párov dvoch reťazcov ako `list` n-tíc `tuple` dvoch reťazcov `str`. Takým spôsobom bolo garantované poradie riadkov v rámci pravej bunky tabuľky a bolo možné ho zvoliť.

Správne rozvrhnutie zobrazovaných údajov do tabuľky je kritické pri vyhodnocovaní výsledkov a je potrebné, aby bolo čo najviac intuitívne. Preto v rámci tejto práce bolo zvolené nasledovné rozvrhnutie:

- podtabuľky reprezentujú jednotlivé procesy spustené pri behu analyzovaného programu,
- riadky v podtabuľkách popisujú práve jeden konkrétny objekt hešu či kľúču,
- v ľavej bunke riadku je typ objektu opísaného v riadku spolu s hodnotou jeho handlu rozšírenou o číslo špecifikujúce poradia medzi objektami s rovnakou hodnotou handlu (príklad: „0x01234567-8“, kde 8 znamená, že ide o deviaty objekt v poradí s daným handlom),
- v pravej bunke riadku sú vypísané zistené hodnoty o danom objekte a jeho využití, ako môže byť napríklad: algoritmus, pôvod daného objektu (volanie, ktorým bol vytvorený), popis využitia – šifrované dáta pri kľúči, vstupný buffer pri objekte hešu.

4.3 Návrh a implementácia testov

V rámci práce boli taktiež navrhnuté a implementované testy funkcionality analyzátora na vrchnej úrovni. Ako súčasť práce bol autorom vytvorený testovací program vykonávajúci všetky v tejto kapitole opísané požadované sledované operácie a detegované chovania programu. Pre otestovanie je nutné najprv vzdialene z funkčného bežiaceho sandboxu Cuckoo získať hrubé nespracované dáta z výstupu analýzy testovacieho programu, ktoré sú následne pred testovaním funkcionality analyzátora spracované Processing modulom Cuckoo, ktorý už môže bežať lokálne na testujúcom stroji. Manuálne generovanie hrubých dát z analýzy je potrebné len v prípade zmeny v testoch alebo v knižnici Cuckoo monitor. Následne je vykonaná analýza modulom Signature. Výsledok tejto analýzy testovacieho programu je výstupný stav analyzátora, ktorý je podrobne automaticky overený podľa požiadaviek zostrojených

podľa špecifík testovacieho programu, ako napríklad v akých formátoch boli jednotlivé kľúče importované, exportované apod. Náhodné dáta ako napríklad hodnoty handle kľúčov, ktoré sú vždy náhodné, sú riešené tak, že výstupom testovacieho programu je aj textový súbor obsahujúci pridelené náhodné hodnoty v danom behu programu. Takto je možné zabezpečiť, že všetky funkcionality navrhnutého analyzátora sú úplné.

5 Experimentálne výsledky

Táto kapitola je venovaná predstaveniu vytvoreného analyzátora, z hľadiska plnenia požadovaných schopností z kapitoly 3. Popísané budú jeho schopnosti detegovať vybrané operácie a vzápätí bude ukázaný spôsob, ktorým je toto chovanie prezentované vo webovom prostredí Cuckoo.

Ďalej bude v tejto kapitole prezentovaná schopnosť analyzovať vzorky reálneho ransomware z minulosti, ktorý zneužíval práve kryptografické prostriedky poskytované knižnicou CryptoAPI. V rámci tejto časti je zároveň posúdená presnosť analýz a jej nedostatky.

Nasledujúca časť je venovaná niektorým zmerateľným štatistikám a v závere kapitoly sú spomenuté zistené poznatky z analýz po nasadení do produkčného prostredia, kde bol vytvorený analyzátor podrobený veľkému množstvu analýz.

5.1 Schopnosti vytvoreného analyzátora

Ako prvé budú predstavené schopnosti interpretovať operácie s objektami hešov CryptoAPI. Potom sú popísané a uvedené všetky implementované schopnosti interpretovať operácie s objektami kľúčov.

5.1.1 Analýza využitia objektov hešov CryptoAPI

Ako bolo už v časti 2.2 opísané, možností na manipuláciu s objektami hešov nie sú veľké. Preto aj prezentácia operácií s objektami hešov je jednoduchá a výstižná. Príkladom výstupu analýzy programu využívajúceho objekty hešov je ukázaný na obrázku 5.1. Hlavička tabuľky reprezentuje názov procesu, ktorý počas analýzy programu daný objekt vytvoril a využíval. Následne v prvom riadku je uvedený prvý z objektov hešov. Tento objekt bol vytvorený ako prázdny objekt hešu volaním funkcie `CryptCreateHash`. Následne bol naplnený dátami, ktoré sú uvedené ako pole nazvané „**Input buffer**“. Pole „**Handle**“ reprezentuje číselný handle, pod ktorým bolo s týmto objektom narábané.

Druhý riadok tabuľky reprezentuje druhý objekt hešu, ktorý vznikol druhým volaním `CryptDuplicateHash`. Pri tomto objekte je v riadku „**Source**“, v ktorom býva uvedené volanie ktorým bol daný objekt vytvorený, uvedený aj „rodičovský“ objekt hešu, z ktorého vznikol. Pri duplikovaní už existujúcich objektov hešu je vstupný buffer hešu naplnený rovnakými dátami ako boli u jeho „rodiča“. Avšak následne sa objekty správajú samostatne a ako je vidno, druhý, duplikovaný objekt, ešte pridával do svojho vstupu dáta navyše oproti jeho „rodičovského“ objektu.

Process: signaturetester.exe (1772)	
Hash 0x0039f160-0	**Handle:** 0x0039f160 **Source:** CryptCreateHash **Algorithm:** SHA-256 **Input buffer:** This\x20is\x20a\x20testing\x20string\x20of\x20only\x20ASCII\x20chars.
Hash 0x0039f0a0-0	**Handle:** 0x0039f0a0 **Source:** CryptDuplicateHash of hash 0x0039f160-0 **Algorithm:** SHA-256 **Input buffer:** This\x20is\x20a\x20testing\x20string\x20of\x20only\x20ASCII\x20chars.This\x20is\x20a\x20testing\x20string\x20of\x20only\x20ASCII\x20chars.

Obr. 5.1: Príklad prezentácie detekcie využitia dvoch objektov hešu CryptoAPI, použitých programom počas analýzy, v tabuľke výstupu analyzačnej signatúry.

5.1.2 Analýza využitia objektov kľúčov CryptoAPI

Knižnica CryptoAPI poskytuje omnoho viac funkcií na manipuláciu s objektami kľúčov ako hešov. Preto aj bolo navrhnutých viac funkcionalít pri analýze operácií s kľúčmi CryptoAPI a aj výsledok analýzy je omnoho verbálnejší ako u prezentácii zistených operácií s hešmi.

Prvým príkladom je základný kľúč, vytvorený lokálne jedným z najjednoduchších volaní funkcie `CryptGenKey`. V ukážke tento kľúč najprv exportoval svoju hodnotu vo formáte `PLAINTEXTKEYBLOB`, následne tento kľúč zašifroval buffer so správou „This is a testing string of only ASCII chars.“. Potom týmto kľúčom bol dešifrovaný šifrový text, ktorého dešifrovaná podoba je zobrazená ako „**Decrypted data**“. Poradie týchto operácii vo výstupe nezodpovedá poradiu, v ktorom boli operácie vykonané, ale je u všetkých objektov kľúčov rovnaké, a to šifrovanie, dešifrovanie a následne exportovanie tohto kľúča. V tomto prípade je možné vidieť pôvodnú hodnotu kľúča, ktorým boli dáta šifrované či dešifrované, pretože bol pri exportovaní použitý formát, kde je hodnota kľúča v čistej podobe (formáty exportu sú bližšie opísané v časti 3.2.3). Ukážka prezentácie rozpoznania takejto aktivity je na obrázku 5.2.

Každý vytvorený objekt kľúča sa objaví v tabuľke a pozostáva z povinných položiek: „**Handle**“, „**Key origin**“, „**Algorithm**“. Položka „**Handle**“, rovnako u hešov, reprezentuje pridelený 32-bitový handle, pod ktorým je s daným objektom operované. Riadok „**Algorithm**“ uvádza algoritmus, ktorého kľúč reprezentuje daný objekt kľúča CryptoAPI. Položka „**Key origin**“ popisuje pôvod príslušného objektu kľúča. Môže sa tam vyskytnúť len napríklad informácia o tom, že daný kľúč bol náhodne vygenerovaný lokálne počas behu, ale aj napríklad, že bol importovaný kľúč, ktorý zatiaľ nebol spozorovaný ako lokálne generovaný.

Process: signaturetester.exe (1772)	
Symmetric key 0x0039f260-0	**Handle:** 0x0039f260 **Key origin:** CryptGenKey **Algorithm:** AES-256 (256 bit) CBC mode **Encrypted data 1:** (Unrecognized data) This\x20is\x20a\x20testing\x20string\x20of\x20only\x20ASCII\x20chars. **Decrypted data 1:** \x08\x02\x00\x00\x10f\x00\x00\x20\x00\x00\x00\x0c\xbc\x82\xc8`)0M\xd5,\xb0\xb6;@\xdc\x8d\xb1\xf1h\x9f\x1c6:\xdb!u\x03\x11\xf9\xe6\xc8\xde **Export 1 as PLAINTEXTKEYBLOB:** \x08\x02\x00\x00\x10f\x00\x00\x20\x00\x00\x00\x0c\xbc\x82\xc8`)0M\xd5,\xb0\xb6;@\xdc\x8d\xb1\xf1h\x9f\x1c6:\xdb!u\x03\x11\xf9\xe6\xc8\xde

Obr. 5.2: Príklad prezentácie detekcie využitia kľúčov CryptoAPI, v tabuľke výstupu analyzačnej signatúry. V tomto prípade ide o kľúč šifry AES-256, pomocou ktorého je raz šifrované, raz dešifrované a zároveň je tento kľúč raz exportovaný. Pole „**Encrypted data**“ predstavuje vstupnú správu do šifrovacej operácie, pole „**Decrypted data**“ zobrazuje výstupnú správu z dešifrovacej operácie.

Taktiež vytvorený analyzátor deteguje opätovné importovanie počas behu exportovaného kľúča a vytvára medzi nimi väzby. Takýto prípad je ukázaný na obrázku 5.3. Zistené spojitosti medzi už v minulosti počas behu programu zachytenými kľúčmi sú prezentované ako prídavná informácia v súčasti informácie o pôvode kľúča „**Key origin**“.

Process: signaturetester.exe (1772)	
Asymmetric key 0x0039f360-0	**Handle:** 0x0039f360 **Key origin:** CryptImportPublicKeyInfo of previous export of key 0x0039f220-0 (RSA Public Key Exchange (1024 bit) CryptGenKey) as Public Key Info **Algorithm:** RSA Public Key Exchange (1024 bit)

Obr. 5.3: Príklad detekcie opätovného využitia kľúčov CryptoAPI exportovaním a importovaním a následná prezentácia ako súčasť poľa „**Key origin**“ vo výstupe analýzy.

Ak je kľúč vytvorený odvodením od hešu, v riadku pre pôvod kľúča je uvedený handle a index špecifikujúci konkrétny heš, z ktorého bol odvodený. Index reprezentuje poradie daného objektu medzi objektami, ktorým bola pridelená rovnaká hodnota handle počas behu programu a býva pripojený za reprezentáciu handlu objektu ako `0xhandle-index`, názvy v tomto formáte sú vždy v ľavom stĺpci popisu

objektu. U každého objektu je tak zaručené, že je kombinácia handle-index unikátna. Podľa tohto je možné nájsť konkrétny heš, z ktorého bol kľúč odvodený a je tak možné vidieť, z akých dát bol vytvorený. Ukážka takto vytvoreného objektu kľúča je na obrázku 5.4.

Process: signaturetester.exe (1772)	
Hash 0x0039f8a0-0	**Handle:** 0x0039f8a0 **Source:** CryptCreateHash **Algorithm:** SHA-256 **Input buffer:** This\x20is\x20a\x20testing\x20string\x20of\x20only\x20ASCII\x20chars.
Symmetric key 0x0039f860-0	**Handle:** 0x0039f860 **Key origin:** CryptDeriveKey from hash object SHA-256 GUID: 0x0039f8a0-0 **Algorithm:** AES-256 (256 bit) CBC mode

Obr. 5.4: Príklad detekcie odvodenia kľúča z výsledku hešu a následná prezentácia ako súčasť poľa „**Key origin**“ popisu kľúča vo výstupe analýzy.

Ako ďalšia užitočná funkcionalita je rozpoznávanie vstupných bufferov do šifrovacích algoritmov pre prípady, kedy sú šifrované nie zdanlivo náhodné dáta. V tejto práci vytvorený analyzátor je schopný rozpoznať nasledujúce druhy bufferu podľa úvodných bajtov. Ide hlavne o schopnosť rozpoznať už v minulosti exportovaného kľúča. Táto funkcionalita urýchľuje rozpoznanie hierarchie kľúčov a umožňuje rýchlo overiť aj analyzovaný program na náchylnosť na útok na nim použitý spôsob šifrovania dát. Ďalšie rozpoznávané typy šifrovaných bufferov sú pravdepodobné užívateľské dáta a pravdepodobné exportované kľúče. Rozpoznávanie šifrovania pravdepodobne súborov na disku vychádza z porovnávania úvodných bajtov s vytvorenými „signatúrami súborov“ (aktuálne implementované signatúry sú dostupné v kóde analyzátora v prílohe C.2.5), ktoré reprezentujú napríklad úvodné bajty spustiteľných súborov, archívov, fotografií a ďalších. Detekcia „pravdepodobného kľúča“ funguje len ako rada či nápoveda, pretože stačí aby úvodným bajtom bufferu bola jedna z vybraných konštánt typu `BLOBu` určujúca jeho štruktúru (konštanty vytýčené v rámci časti 3.2.3). Všetky z vyššie opísaných schopností sú ukázané v obrázku 5.5.

Process: signaturetester.exe (1772)	
Symmetric key 0x0039f260-0	**Handle:** 0x0039f260 **Key origin:** CryptGenKey **Algorithm:** AES-256 (256 bit) CBC mode **Encrypted data 1:** (Unrecognized data) This\x20is\x20a\x20testing\x20string\x20of\x20only\x20ASCII\x20chars. **Encrypted data 2:** (Unrecognized likely PRIVATEKEYBLOB) \x07\x02\x00\x00\x00\xa4\x00\x00RSA2\x00\x04\x00\x00\x01\x00\x01\x00\x15\x80\xa8\xa2\x87\xce\xef\x19\xdf\xb5\xeb\xc5%M\xf4\x82I\xd6\xd6\x83\xc7\xb8\x03\1`1\x1b\x9cn8\xb8\x02\x0a4\x14J\xf3\x02\xa9\xcf\x98N8\xd6W\x95\xd5\x11\xff,\xa8\xbaz,\x1c\x90V\xef\xb0s\xb1l\xcbM4\x0fs\x00\xc2\xf9Ye\x19\xaa\xee\xa8o\xa3'\xae4\xa6:\xb3\xc9\x17\xe4\xe3\xb5\x09\x19V\x8b\xb4%\xb0\x95$\xd9va\x88\xe7\xc68\x06\x83\x8aO\x8c?+Z\xca\xe0=\xf4\xcd\xac\x0b\xdfc\x1f\xae\x91\x89\xbfK\xe9\xc7.Cno9\xae\xe0\xdd&\xd7\xf9\xf6\xdf\xd5gw\x11\x (1251 more characters) **Encrypted data 3:** (Likely PE file) MZThis\x20should\x20match\x20a\x20PE\x20file\x20signature **Encrypted data 4:** (Exported AES-256 (256 bit) key 0x0039f260-0 exported as PLAINTEXTKEYBLOB, used 6 times, from CryptGenKey) \x08\x02\x00\x00\x10f\x00\x00\x20\x00\x00\x00\x0c\xbc\x82\xc8`)0M\xd5,\xb0\xb6;@\xdc\x8d\xb1\xf1h\x9f\x1c6:\xdb!u\x03\x11\xf9\xe6\xc8\xde **Export 1 as PLAINTEXTKEYBLOB:** \x08\x02\x00\x00\x10f\x00\x00\x20\x00\x00\x00\x0c\xbc\x82\xc8`)0M\xd5,\xb0\xb6;@\xdc\x8d\xb1\xf1h\x9f\x1c6:\xdb!u\x03\x11\xf9\xe6\xc8\xde

Obr. 5.5: Ukážka prezentácie detekovaného špeciálneho typu šifrovaného bufferu, všetky tri implementované rozpoznávané typy. Prvý buffer „**Encrypted data 1**“ predstavuje šifrovanie nerozpoznaných dát. Druhý šifrovaný buffer reprezentuje pravdepodobný exportovaný kľúč na základe úvodnej sekvencie bajtov `0x07 0x02`. Tretí buffer upozorňuje na detekciu „signatúry súboru“, teda pravdepodobný súbor PE podľa úvodných bajtov `MZ`. Pri štvrtom šifrovanom bufferi je detegovaný v minulosti už zachytený exportovaný kľúč vo formáte `PLAINTEXTKEYBLOB` a je tam zároveň pridaný odkaz naň (v tomto prípade je to ten istý kľúč).

5.2 Ukážka schopností analyzátora pri analýze reálneho malware

V tejto kapitole budú predstavené príklady výstupu analýzy pri vzorkách malwaru. Ukázané budú analýzy ransomwaru, pretože tie vykonávajú veľké množstvo operácií využívajúcich prostriedky kryptografie a ich analýza na základe hrubého nespracovaného behaviorálneho logu zo sandboxu by vyžadovala niekoľko násobne viac času. Porovnané budú informácie, ktoré je analytik schopný zistiť z výstupu automatickej analýzy oproti informáciám o využití kryptografie pri podrobnej manuálnej analýze, ktorá je o niekoľko rádov časovo náročnejšia.

Ransomware Alcatraz Locker

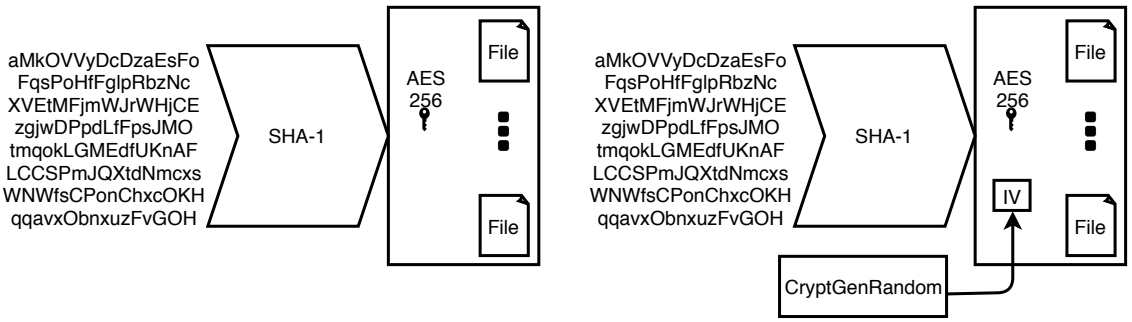
Jednoduchým príkladom je ransomware Alcatraz Locker. Z výstupu automatickej analýzy (dostupnej aj ako HTML v prílohe na CD C.2.2) je možné zistiť jednoduchú schému šifrovania, ukázanú ako diagram na sebe závislých kľúčov na obrázku 5.6 (blok s názvom algoritmu predstavuje objekt kľúču ktorým sú šifrované dáta blokov v ňom zahrnuté). Je vidno použitie symetrickej šifry AES-256 ako „hlavného kľúča“, ktorý je odvodený z hešu SHA-1 zdanlivo náhodného reťazca z písmen abecedy v ASCII dĺžky 128 znakov. Z tohto je jasne viditeľná slabina tohto ransomwaru. Používa na všetky súbory jeden identický kľúč a tento kľúč je generovaný z hešu.

Na základe výstupu automatickej analýzy je tak možné začať pátrať po pôvode dát použitých na odvodenie kľúča. Po vyhľadaní pôvodu dát hešu, z ktorého bol kľúč odvodený, je možné zistiť, že dáta použité ako kľúč nie sú úplne náhodné, ale sú odvodené z aktuálneho času, a je tak možné na použitú schému zaútočiť. Aj tomu tak v minulosti bolo, na tento ransomware bol vydaný dekryptor [27].

Nedostatkom tejto automatickej analýzy je neschopnosť zistiť použité hodnoty inicializačného vektora, pretože v hooku dostupné rozhranie neumožňuje hodnotu inicializačného vektoru bezpečne a spoľahlivo získať a zaznamenať (dôvod je bližšie opísaný v časti 3.2.3).

Ransomware Spora

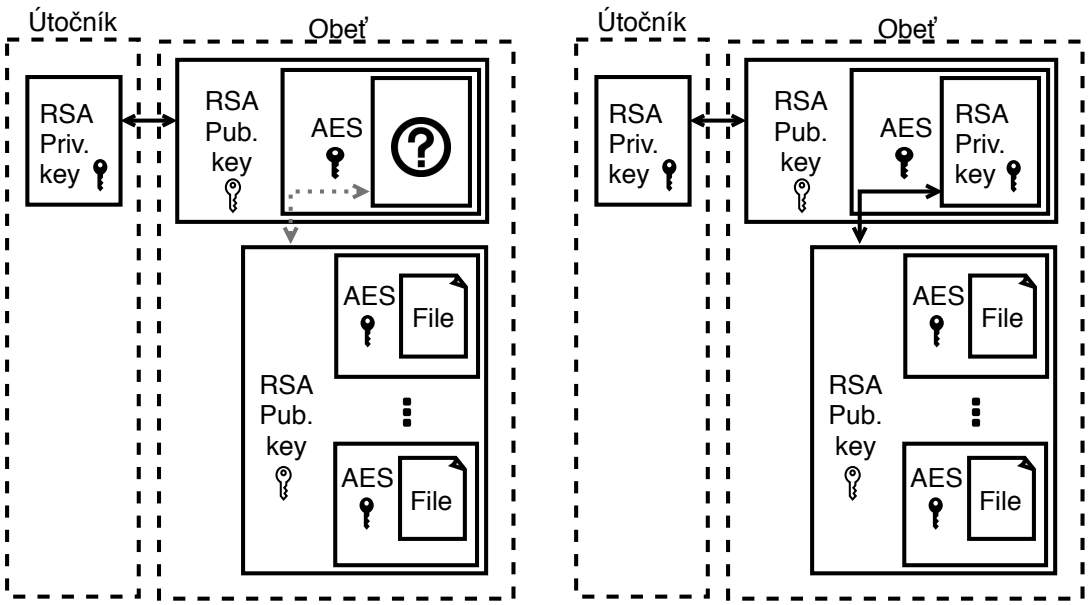
Druhým príkladom demonštrácie schopností analýzy v tejto práci vytvoreného analyzátora je ransomware Spora. U tohto ransomwaru je použitá až príliš obskurná schéma šifrovania užívateľských dát, ktorú je náročné a pracné zistiť na základe statickej analýzy a aj z výstupu dynamickej analýzy v Cuckoo pomocou manuálnej analýzy hrubého výstupu Behavioral log. Podľa slov analytika zo spoločnosti Avast, trvala analýza celej kryptografickej schémy ransomwaru Spora až 3-4 dni manuálnej analýzy na základe sandboxu a analýzy strojového kódu.



Obr. 5.6: Porovnanie schémy odvoditeľnej z tabuľky výstupu analýzy v práci vytvoreným analyzátorom (vľavo) a schémy rozpoznateľnej pri podrobnej manuálnej analýze (vpravo) – Alcatraz Locker

Analýza použitia kryptografie u tohto ransomwaru analyzátorom bola úspešná v celom rozsahu aktivity ransomwaru. Z výsledku tejto analýzy (taktiež dostupného ako HTML v prílohe na CD C.2.2) je možné zistiť schému šifrovania súborov a následne kľúčov ako je zobrazená na obrázku 5.7. Je vidno, že väčšina závislostí kľúča na druhý kľúč bola analyzátorom zachytená a rozpoznaná. Jedinou takouto väzbou, ktorý analyzátor nerozpoznal, bola väzba exportovaného lokálne náhodne vygenerovaného súkromného kľúča šifry RSA. Nastalo to z dôvodu, že analyzátor rozpoznáva šifrovanie exportovaných kľúčov len v nepozmenenej podobe, tj. šifrovaný kľúč musí byť identický s podobou, akou bol exportovaný. Avšak ransomware Spora s týmto kľúčom manipuloval a preto nebol detegovaný ako vstup do šifry AES v hornom bloku. Tento kľúč bol, po exportovaní vo formáte `PRIVATEKEYBLOB`, zakódovaný kódovaním Base64 a ešte následne za naň boli pridané dáta použité útočníkom na identifikáciu obete.

Napriek tomuto nedostatku, na ktorého rozlúštenie je požadovaná analýza pôvodu vstupu do vyššieho bloku AES analytikom s intuíciou, je stále možné považovať analýzu za úspešnú, pretože všetky informácie dostupné pre analyzátor boli interpretované správne. Stále však aj takáto analýza urýchľuje celkovú analýzu tým, že odvedie pospájanie spolu súvisiacich volaní CryptoAPI dokopy a umožní sústrediť sa na iné detaily.



Obr. 5.7: Porovnanie schémy šifrovania odvoditeľnej z tabuľky výstupu vytvoreného analyzátora (vľavo) a skutočnej schémy zistenej manuálnou analýzou (vpravo) – Spora

5.3 Štatistiky

V rámci tejto časti budú predstavené niektoré zmerané veličiny vytvoreného analyzátora. Prvou uvedenou zmeranou metrikou je hodnotenie výkonu z hľadiska doby trvania analýzy a odmeraná penalizácia za pridanú novú signatúru. V druhej časti je posúdená úspešnosť a spoľahlivosť analyzátora.

5.3.1 Záťažový test na dopad na dĺžku analýzy

Porovnané boli dĺžky vykonávania celého Signature modulu Cuckoo (pozostávajúceho s viacerých signatúr) nad rovnakým setom volaní a parametrov z behu ransomware Rapid, najprv bez v tejto práci vytvoreného analyzátora a následne spolu s ním. Tento ransomware bol zvolený ako program na záťažové testovanie preto, lebo počas jeho behu býva použité veľké množstvo volaní CryptoAPI, a pretože počas celej dĺžky analýzy intenzívne šifruje dáta. Konkrétne pri jednom opakovaní tohto merania bolo zanalyzovaných celkom 44806 volaní CryptoAPI knižnice, ktoré tento ransomware vykonal počas doby jeho analýzy. Získané výsledky sú uvedené v tabuľke 5.1.

Uvedené hodnoty boli získané zmeraním dĺžky 20 behov celého modulu Signature pre každý prípad na počítači s CPU Intel Xeon E3-1240 v5 pri 3,5 GHz v interpretéri

Tab. 5.1: Doba trvania behu celého modulu Signature sandboxu Cuckoo počas 20 behov.

Doba behu modulu Signature	Stredná hodnota	Medián	Št. odchýlka
Bez analyzátora CryptoAPI	17,530 s	17,532 s	0,085 s
S analyzátorom CryptoAPI	18,514 s	18,484 s	0,168 s
Relatívny nárast	+5, 613 %	+5, 430 %	+97, 65 %

Python 2.7.15. Na základe týchto údajov je možné považovať dopad na dĺžku analýzy za zanedbateľný, pretože dopad na samotný modul Signature je v priemere 5,61 %, pričom trvanie modulu Signature zaberá v porovnaní s celou dĺžkou analýzy len malú časť (celková dĺžka analýzy sa pohybuje rádovo v oblasti jednotiek minút). Taktiež je dôležité poznamenať, že táto penalizácia je relevantná len pri analýzach programov, ktoré využívajú veľké množstvo volaní knižnice CryptoAPI. U ostatných programov, ktoré nepoužívajú knižnicu CryptoAPI vôbec, je cena analyzátora nulová, inak rastie lineárne s počtom analyzovaných volaní funkcií knižnice CryptoAPI.

5.3.2 Úspešnosť analýz a spoľahlivosť

Následne bola vykonaná skúška a vyhodnotenie spoľahlivosti analyzátora. Bolo zvolené obdobie jeden mesiac a následne boli za toto obdobie pozberané údaje o počte úspešných analýz a počte chybných analýz. Za účelom zistenia počtu chybných analýz bola vytvorená ďalšia signatúra v module Signature sandboxu, ktorá bola prihlásená ako detegovaná len v prípade, že nastala neošetrená výnimka v tele hlavnej analyzujúcej signatúry. Následne tak bolo možné sledovať hlásenia neúspešných analýz analyzačnou signatúrou ako úspešné detekcie signatúry reprezentujúcej pád analyzačnej. Za sledované obdobie jedného mesiaca bola zistená úspešnosť 99,374 %, ktorú možno považovať za úspešnú. Neúspešné analýzy zlyhávali pri ojedinelých nepokrytých krajných prípadoch, ktoré neboli v rámci tejto práce preskúmané a pokryté. Počas tejto práce bolo vytvorených viacero verzií analyzátora s opravami stability a finálna verzia obsahuje viacero opráv stability oproti predchádzajúcim, avšak na opravu posledných zostávajúcich chýb by bolo potrebné analyzovať vnútorný kód a spôsob fungovania knižnice CryptoAPI. V rámci tejto práce však bolo s knižnicou CryptoAPI narábané ako s čiernou skrinkou a jej chovanie v niektorých prípadoch sa nepodarilo ozrejmiť. U všetkých z ostávajúcich chýb nastali chyby pri interpretovaní volaní funkcií, ktoré predpokladajú určité predchádzajúce akcie, ktoré neboli vykonané. Tak sa do analyzátora dostalo, podľa CryptoAPI, úspešné volanie, ktoré by podľa analyzátora nemalo uspieť.

5.4 Analýza výsledkov získaných automatickými analýzami

Počas tvorby tejto práce bola tiež vykonaná revízia výsledkov získaných automatických analýz vytvoreného analyzátora. Takto boli objavené nové pôvodne nenavrhované schopnosti analyzátora. Revízia bola uskutočnená vytvorením zhlukov sebe vzájomne podobných spustiteľných programov, z ktorých následne boli náhodne vybrané reprezentujúce programy. Tie boli poslané na analýzu a ich výsledky boli automaticky pozberané. Následne boli tieto výsledky analýz programov reprezentujúcich podobné zhluky analyzované.

Bolo zistené, že analyzátor je schopný analyzovať využitie kryptografie aplikácií .NET, ktoré využívajú knižnicu `System.Security.Cryptography`. Pri bližšej inšpekcii daných programov bolo potvrdené, že algoritmus RSA v knižnici `System.Security.Cryptography` je vnútorne poskytovaný triedou `RSACryptoServiceProvider`, ktorá využíva knižnicu CryptoAPI. Aplikácie .NET využívajúce algoritmus RSA poskytovaný touto systémovou knižnicou sú tak tiež subjektom na analýzu spôsobu využitia tohto algoritmu.

Aj na základe týchto zistení bola vyslovená hypotéza o možnosti extrahovať z vytvoreného analyzátora kľúče RSA importované zo spustiteľného súboru či inak externe distribuované (tj. kľúče, ktoré neboli generované lokálne) za účelom poskytnutia unikátnej vlastnosti, ktorá by spájala súvisiace programy na základe toho, že by mali v sebe obsiahnutý spoločný RSA kľúč, síce by sa ich strojový kód líšil. Na základe tejto spoločnej vlastnosti by následne mohol byť vytvorený nový systém na klasifikáciu malware. Táto funkcionalita bola pridaná do funkcionality analyzátora, ktorý začal zberať verejné RSA kľúče, ktoré neboli lokálne vytvorené, a normalizoval ich do spoločnej formy. Táto forma bola určená ako heš SHA-256 z kľúču reprezentovaného binárne vo formáte `RSAPublicKey` zo štandardu PKCS#1 kódovanom v kódovaní DER (tento formát je opísaný v časti 2.3). Táto forma bola po konzultácii zvolená kvôli tomu, pretože nebolo potrebné vedieť konkrétnu hodnotu kľúču RSA, ktorý býva dlhý, a tiež za účelom ľahšej manipulácie v prípade, že by boli tieto hodnoty používané na tvorbu pravidiel na zhlukovanie súvisiacich programov (ako napríklad rôzne verzie malware od rovnakého autora s rovnakým RSA kľúčom).

Následne boli zberané výsledky analýz, kde nastalo importovanie takýchto verejných RSA kľúčov. Potom boli analyzované súbory pozhlukované podľa spoločných importovaných verejných RSA kľúčov a boli vyhodnotené obsahy takto vytvorených zhlukov. Takto bolo vyhodnotené obdobie 75 hodín. Počas tohto obdobia bolo získaných 785 unikátnych RSA kľúčov. Najväčší zhluk pozostával zo 2767 súborov, z ktorých bolo 926 označených ako čisté programy bez zlých úmyslov. Aj v ďal-

ších veľkých zhlukoch o veľkostiach medzi 1500 až 500 súborov boli veľké množstvá, o veľkosti až okolo 10-30 % neškodných súborov.

Po vyhodnotení takto vytvorených zhlukov za takéto obdobie nebolo možné vysloviť, že by takéto zhlukovanie bolo užitočné na vytváranie zhlukov programov za účelom akcelerácie klasifikovania nových vzoriek malware. Avšak nie je vylúčené, že by sa výsledky mohli líšiť, ak by bolo sledované obdobie dostatočne dlhé, aby sa objavilo väčšie množstvo generácii a verzií programov. Takto bolo rozhodnuté hlavne na základe toho, že väčšie množstvo zhlukov obsahovalo veľké množstvo rôzne klasifikovaných súborov a len vo výnimočných prípadoch vznikli zhluky klasifikované ako jedna rodina. Z dôvodu veľkej nepredvídateľnosti obsahu zhlukov by nebolo možné spoľahlivo určiť klasifikáciu súboru len na základe ním importovaného verejného RSA kľúča, ktorý by bol importovaný aj iným programom.

6 Záver

V tejto práci bola predstavená knižnica distribuovaná so systémami Microsoft Windows, a v súčasnosti naďalej používaná, Microsoft CryptoAPI. Boli preskúmané jej schopnosti a možnosti využitia. Tieto funkcie bývajú zneužívané autormi malware na páchanie škôd obetiam a na zadržiavanie súborov obetí ako zámienku na zaplatenie výkupného. Podľa postupov nutných pri analýze vzorky takéhoto malware využívajúcej túto knižnicu bol navrhnutý spôsob, ktorým je možné interpretovať aktivitu pomocou sledovaných volaní funkcií CryptoAPI. Táto práca bola vykonaná v spolupráci so spoločnosťou Avast. Na zachytávanie a sledovanie volaní funkcií analyzovanými programami býva v spoločnosti Avast používaný nástroj na dynamickú analýzu, sandbox Cuckoo. Tento nástroj aj poskytuje bohaté možnosti na tvorbu vlastných rozšírení. Keďže interpretácia operácií s knižnicou CryptoAPI a ich analýza je z veľkej časti automatizovateľná, bol zvolený cieľ vytvoriť rozšírenie sandboxu Cuckoo.

Sandbox Cuckoo je schopný automaticky analyzovať programy a zbierať informácie o ich chovaní počas behu. Na extrahovanie informácií o využití knižnice CryptoAPI sú využité možnosti poskytované samotným sandboxom. Ten bol rozšírený o nové tzv. hooky, ktoré slúžia na presmerovanie pôvodných volaní funkcií na vlastné funkcie, v ktorých je možné analyzovať a zaznamenávať parametre, s ktorými bola funkcia zavolaná. Keďže knižnica CryptoAPI poskytuje svoju funkcionalitu prostredníctvo dynamicky linkovaných knižníc, bolo možné zachytávať vybrané volania knižnice CryptoAPI skúmaným programom a získať z nich potrebné informácie. Takto získané informácie sú následne ďalej analyzované v sandboxe, kde sú postupne predané aj v tejto práci navrhnutému analyzačnému modulu, určeného na interpretáciu operácií s knižnicou CryptoAPI.

Tento analyzačný modul bol vytvorený ako signatúra, ktorá prijíma informácie o zaznamenaných volaniach funkcií CryptoAPI. Signatúra v sebe vo vnútri napodobňuje chovanie CryptoAPI a naplňuje vnútornú štruktúru, ktorá je následne na konci analýzy prezentovaná ako tabuľka zhrňujúca aktivitu s CryptoAPI analyzovaným programom. Výsledná tabuľka je integrovaná ako súčasť výsledku analýzy programu vo webovom prostredí sandboxu Cuckoo. Sú v nej prezentované vytvorené objekty CryptoAPI a informácie o ich využití. Taktiež je detegovaná aktivita, ktorá sa týka manipulácie s exportovanými kľúčmi, ktoré plnia vysokú úlohu napríklad u ransomware. V prípade ransomware, je možné vďaka prezentovanému výsledku analýzy použitia CryptoAPI určiť spôsob narábania s kľúčmi použitými na šifrovanie súborov, a či sú šifrované inými kľúčmi. Taktiež je možné identifikovať použitú zraniteľnú schému šifrovania súborov, a tak rozpoznať prípadne nesprávnym spôsobom použité kryptografické prostriedky. V takom prípade je možné vydať dekryptor pre tento

ransomware.

Navrhnutý a vytvorený analyzátor bol úspešne implementovaný v celom rozsahu stanovenom pri jeho návrhu. Následne bol nasadený do produkčného prostredia spoločnosti Avast, kde bol podrobený množstvu analýz. Boli takto získané údaje o jeho úspešnosti a presnosti, ktorá bola v rámci tejto práce preverená. Dosiahnutá úspešnosť analýz za sledované jednomesačné obdobie dosiahla 99,374 %. Taktiež bola vyhodnotená jeho výpočetná náročnosť, ktorá sa predstavovala približne 5,61 % predĺženia trvania jednej etapy analýzy. Tento nárast dĺžky časti analýzy je vzhľadom na celkovú dĺžku jednej analýzy zanedbateľný a tak je možné analyzátor hodnotiť ako úspešný. Práve naopak, bolo dosiahnuté značné urýchlenie prvotnej analýzy nových kmeňov malware, ktorej dĺžka klesla z rádov dní (manuálnej analýzy) na jednotky minút. To môže výrazne pomôcť s odhalením slabín týchto kmeňov a pomôcť infikovaným obetiam.

Literatúra

[1] IOT ANALYTICS. *State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating*. [online]. 2018, posledná aktualizácia 8. 8. 2018 [cit. 14. 12. 2018]. Dostupné z URL: <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>

[2] MOSER A., Kruegler C., Kirda E. Limits of Static Analysis for Malware Detection. In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. [online]. Miami Beach, USA: IEEE, 2. 1. 2008. s. 421 - 430. ISSN 1063-9527. Dostupné z: <https://ieeexplore.ieee.org/abstract/document/4413008/metrics#metrics>

[3] AV-TEST. *Security Report 2017/18*. [online]. 2018, posledná aktualizácia 20. 7. 2018 [cit. 4. 12. 2018]. Dostupné z URL: <https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2017-2018.pdf>

[4] AV-TEST. *Security Report 2015/16*. [online]. 2016 [cit. 4. 12. 2018]. Dostupné z URL: <https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2015-2016.pdf>

[5] EILAM, Eldad. *Reversing secrets of reverse engineering*. Indianapolis: Wiley, 2005. xxviii, 589 s. ISBN 0-7645-7481-7.

[6] O'Gorma G., McDonald Geoff. *Ransomware: A Growin Menace*. [online]. 2012, Symantec Corporation [cit. 14. 12. 2018]. Dostupné z URL: <http://www.01net.it/whitepaper_library/Symantec_Ransomware_Growing_Menace.pdf>

[7] CHEBYSHEV V., SINITSYN F., PARINOV D., LISKIN A., KUPREEV O. *IT threat evolution Q2 2018. Statistics*. [online]. Kaspersky lab, 2018, posledná aktualizácia 6. 8. 2018 [cit. 4.12.2018] Dostupné z: <https://securelist.com/it-threat-evolution-q2-2018-statistics/87170/>

[8] MILKOVIČ, Marek. *Systém pro detekci vzorů v binárních souborech*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Matula Peter.

[9] LIU K., TAN H.B.K., CHEN X.: Binary Code Analysis. *Computer*. [online]. Volume: 46, Issue: 8. IEEE, 20. 8. 2013, 128 s., [cit. 24. 11. 2018]. ISSN 0018-9162. Dostupné z: <https://ieeexplore.ieee.org/abstract/document/6583187/metrics#metrics>

[10] Cuckoo Foundation. *Cuckoo sandbox – Automated malware analysis*. [online]. Dostupné z URL: <https://cuckoosandbox.org/>

[11] Optiv, Inc. *cuckoo-modified*. [online]. posledná aktualizácia 6. 8. 2018 [cit. 4.12.2018] Dostupné z URL: <https://github.com/brad-accuvant/cuckoo-modified>

[12] spender-sandbox. *cuckoo-modified*. [online]. posledná aktualizácia 6. 8. 2018 [cit. 4.12.2018] Dostupné z URL: <https://github.com/spender-sandbox/cuckoo-modified>

[13] Spengler Brad. *Improving Reliability of Sandbox Results*. [online]. 2014, posledná aktualizácia 4. 11. 2014 [cit. 14.12.2018] Dostupné z URL: <https://www.optiv.com/blog/improving-reliability-of-sandbox-results>

[14] MENEZES, Alfred J, Paul C.van OORSCHOT a Scott A VANSTONE. *Handbook of applied cryptography*. Boca Raton: CRC Press, 1997, 780 s. ISBN 0-8493-8523-7.

[15] MICROSOFT. *MSDN: CSPs and the Cryptography Process*. [online]. posledná aktualizácia 31. 5. 2018 [cit. 4. 12. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/desktop/seccrypto/csps-and-the-cryptography-process>

[16] MICROSOFT. *MSDN: Cryptography API: Next Generation*. [online]. posledná aktualizácia 31. 5. 2018 [cit. 4. 12. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/desktop/seccng/cng-portal>

[17] QUALYS SSL LABS. *SSL and TLS Deployment Best Practices*. [online]. posledná aktualizácia 30. 5. 2017 [cit. 4. 12. 2018]. Dostupné z URL: <https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices>

[18] MICROSOFT. *MSDN: Cryptographic Provider Types*. [online]. posledná aktualizácia 31. 5. 2018 [cit. 4. 12. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/desktop/seccrypto/cryptographic-provider-types>

[19] MICROSOFT. *MSDN: CryptExportKey function*. [online]. posledná aktualizácia 9. 11. 2018 [cit. 4. 12. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/desktop/api/wincrypt/nf-wincrypt-cryptexportkey>

[20] MICROSOFT. *MSDN: Data Hashes*. [online]. posledná aktualizácia 31. 5. 2018 [cit. 14. 12. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/desktop/seccrypto/data-hashes>

[21] JONSSON, J., KALISKI B. *RFC 3447 – Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. [online]. IETF, 2003. [cit. 24.11.2018]. Dostupné z: <https://tools.ietf.org/html/rfc3447>

[22] *ITU-T X.680 – Abstract Syntax Notation One (ASN.1): Specification of basic notation*. [online]. [cit. 14. 12. 2018]. Dostupné z URL: <https://www.itu.int/itu-t/recommendations/rec.aspx?rec=X.680>.

[23] *ITU-T X.690 – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. [online]. [cit. 14. 12. 2018]. Dostupné z URL: <https://itu.int/ITU-T/X.690>.

[24] MICROSOFT. *MSDN: CryptGetHashParam function*. [online]. posledná aktualizácia 12. 5. 2018 [cit. 14. 12. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/desktop/api/Wincrypt/nf-wincrypt-cryptgethashparam>

[25] MICROSOFT. *MSDN: CryptDeriveKey function*. [online]. posledná aktualizácia 12. 5. 2018 [cit. 14. 12. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/desktop/api/wincrypt/nf-wincrypt-cryptderivekey>

[26] MICROSOFT. *MSDN: BLOBHEADER structure*. [online]. posledná aktualizácia 12. 5. 2018 [cit. 14. 12. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/desktop/api/wincrypt/ns-wincrypt-_publickeystruc>

[27] Křoustek J. *Avast releases four free ransomware decryptors*. 2016, Avast software [online]. posledná aktualizácia 1. 12. 2016 [cit. 10. 5 2019]. Dostupné z URL: <https://blog.avast.com/avast-releases-four-free-ransomware-decryptors>

Zoznam symbolov, veličín a skratiek

3DES	Triple DES (Data Encryption Standard)
AES	Advanced Encryption Standard
ASN.1	Abstract Syntax Notation One
CNG	Cryptography API: Next Generation
CSP	cryptographic service provider
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DLL	Dynamicky linkovaná knižnica
PKCS#1	Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications

Zoznam príloh

A Používané konštanty **62**
A.1 Delenie volaní hešov a kľúčov CryptoAPI 62
A.2 Konštanty operačných módov blokovej šifry 63
A.3 Konštanty používané pri importovaní kľúčov CryptoAPI 63

B Zjednodušený prehľad hookovaných volaní CryptoAPI a zaznamenávaných parametrov **64**

C Dokumentácia priloženého CD **65**
C.1 Obsah CD 65
C.2 Dokumentácia k priloženému obsahu 65
C.2.1 Definície hookov 65
C.2.2 Ukážky výsledkov analýz v HTML 66
C.2.3 Program na testovanie všetkých operácií s CryptoAPI 66
C.2.4 Testovací skript na overenie funkcionalít analyzátora 66
C.2.5 Zdrojový kód analyzátora 66

A Používané konštanty

Táto príloha obsahuje vybrané v texte používané konštanty a ich hodnoty.

A.1 Delenie volaní hešov a kľúčov CryptoAPI

V práci sú viackrát delené funkcie na volania týkajúce sa operáciám s kľúčmi a hešmi. Nasledujúci výpis uvádza zoznamy obsahu spomínaných kategórií volaní.

Výpis A.1: Rozdelenie API volaní na volania týkajúce sa hešov a kľúčo v rámci tejto práce

```
hash_api_calls = {
  'CryptHashData',
  'CryptGetHashParam',
  'CryptCreateHash',
  'CryptDuplicateHash',
  'CryptDestroyHash'
}
key_api_calls  = {
  'CryptEncrypt',
  'CryptImportKey',
  'CryptDecrypt',
  'CryptDestroyKey',
  'CryptSetKeyParam',
  'CryptGenKey',
  'CryptDuplicateKey',
  'CryptExportPublicKeyInfo',
  'CryptImportPublicKeyInfo',
  'CryptDeriveKey',
  'CryptExportKey',
  'CryptGetUserKey',
}
```

A.2 Konštanty operačných módov blokovej šifry

Výpis A.2: Konštanty predávané ako parameter pri zmene operačného módu šifry vo volaní `CryptSetKeyParam`

```
cipher_modes = {
    1: 'CBC',
    2: 'ECB',
    3: 'OFB',
    4: 'CFB',
    5: 'CTS',
}
```

A.3 Konštanty používané pri importovaní kľúčov CryptoAPI

Výpis A.3: Všetky konštanty používané na identifikáciu typu `BLOBu`

```
blob_types = {
    0x1: "SIMPLEBLOB",
    0x6: "PUBLICKEYBLOB",
    0x7: "PRIVATEKEYBLOB",
    0x8: "PLAINTEXTKEYBLOB",
    0x9: "OPAQUEKEYBLOB",
    0xa: "PUBLICKEYBLOBEX",
    0xb: "SYMMETRICWRAPKEYBLOB",
    0xc: "KEYSTATEBLOB",
}
```

B Zjednodušený prehľad hookovaných volaní CryptoAPI a zaznamenávaných parametrov

Táto príloha obsahuje príklad a vysvetlenie formátu, v ktorom sú uvedené definície hookov v súbore `definicie_hookov.txt` na priloženom CD, ktorého obsah je popísaný v prílohe C.

Výpis B.1: Formát zápisu v nasledujúcich výpisoch

```
<názov volania>
  <datatyp parametru 1> <názov parametru 1>
  <datatyp parametru 2> <názov parametru 2>
  <datatyp parametru ...> <názov parametru ...>
  <datatyp parametru n> <názov parametru n>
> LOG <prípadná podmienka>: <formátovací reťazec>, <názov
   ↪ parametru v Cuckoo>, <logovaný parameter x>, <názov
   ↪ parametru v Cuckoo>, <logovaný parameter y>, ... (podľa
   ↪ znakov v formátovacom reťazci)
  // výstupné mapovanie vstupu do hooku a
  // zaznamenané údaje dostupné v Cuckoo
  <datatyp parametru x> <logovaný parameter x> -> <výsledný
     ↪ datatyp v Cuckoo> <výsledný názov parametru v Cuckoo>
  <datatyp parametru y> <logovaný parameter y> -> <výsledný
     ↪ datatyp v Cuckoo> <výsledný názov parametru v Cuckoo>
  ...
```

Výpis B.2: Ukážka transformácie hooku funkcie `CryptCreateHash`, predstaveného vo výpise 1.1, do zjednodušeného formátu definovaného vo výpise B.1.

```
CryptCreateHash
  HCRYPTPROV hProv
  ALG_ID Algid,
  HCRYPTKEY hKey
  DWORD dwFlags
  HCRYPTHASH *phHash
> LOG: "phpiP", "hProv", hProv, "Algid", Algid, "hKey", hKey,
   ↪ "Flags", dwFlags, "hHash", phHash
  void* hProv -> str 'hProv'
  void* Algid -> str 'Algid'
  void* hKey -> str 'hKey'
  DWORD dwFlags -> int 'Flags'
  void** phHash -> str 'hHash' (hodnota na adrese)
```

C Dokumentácia priloženého CD

V tejto prílohe je najprv popísaný obsah priloženého CD a následne je uvedená dokumentácia k všetkým obsiahnutým súborom či programom.

C.1 Obsah CD

Na priloženom CD sa nachádzajú na vrchnej úrovni dva súbory. Okrem digitálnej kópie tejto práce `bp_micka.pdf` sa na nej nachádza archív `prilohy.zip`, ktorého obsah a štruktúru je možné vidieť nižšie v ukážke.

```
/bp_micka.pdf ..... digitálna kópia práce
/prilohy.zip ..... vrchný adresár archívu na priloženom CD
    definicie_hookov.txt ..... ostanté definície hookov vo formáte z prílohy B
    ukazky ..... adresár s ukážkami výsledkov analýz v HTML z kapitoly 5.2
        spora
            Cuckoo Sandbox.htm
        alcatraz
            Cuckoo Sandbox.htm
        tester
            Cuckoo Sandbox.htm
    tester ..... zdrojový kód testovacieho programu na operácie CryptoAPI
        SignatureTest.sln
        SignatureTest.exe
    testscript ..... zdrojový kód testovacieho skriptu
        cryptoapi_anlaysis_tests.py
    analyzator ..... finálna verzia zdrojového kódu analyzátora
        crypto_api_utils
            constants.py
        crypto_api_analysis.py
```

C.2 Dokumentácia k priloženému obsahu

V tejto časti sú opísané jednotlivé adresáre obsiahnuté v archíve `prilohy.zip`. Po rozbalení archívu sú dostupné offline ukážky výsledku analýz a zdrojové kódy programov, ktoré boli vytvorené počas tejto práce.

C.2.1 Definície hookov

V súbore `definicie_hookov.txt`, vo vrchnom adresári archívu, sa nachádzajú všetky používané hooky v zjednodušenom formáte. Je v ňom možné zistiť názvy a typy parametrov, s ktorými následne analyzátor pracuje. Použitý formát je vysvetlený v prílohe B.

C.2.2 Ukážky výsledkov analýz v HTML

Súčasťou prílohy sú dva výsledky analýz, ktoré sú opísané v časti 5.2 a jedna analýza testera, ktorého zdrojový kód je taktiež súčasťou prílohy v adresári `tester`. V adresári `ukazky` sa nachádzajú tri adresáre `alcatraz`, `spora` a `tester`. V každom z nich sa nachádza súbor nazvaný `Cuckoo Sandbox.htm`. Tento súbor obsahuje výňatok z HTML výsledku analýzy. V rámci týchto súborov sú všetky odkazy na zdroje, ako štýl CSS a obrázky, upravené na relatívnu cestu do zložky `Cuckoo Sandbox_subory`, ktorá sa nachádza v adresári pri samotnom HTML dokumente. Takto by malo byť možné tieto súbory zobraziť lokálne v prehliadači, buďto pomocou označenia súbora a zvolením otvorenie v prehliadači, alebo pomocou URL v adresovom riadku prehliadača vo forme `file:///<cela_cesta_k_suboru>/Cuckoo Sandbox.htm`. Správne zobrazenie je overené v prehliadačoch Google Chrome 74 a Mozilla Firefox 66.

C.2.3 Program na testovanie všetkých operácií s CryptoAPI

V adresári `tester` sa nachádza zdrojový kód programu, ktorý je používaný v testoch v tejto práci vytvoreného analyzátora. Úlohou tohto programu je vykonať všetky operácie, ktoré musí analyzátor zvládnuť zanalyzovať. Vykonáva tak všetky v časti 3 vybrané operácie ponúkané knižnicou CryptoAPI, ktoré je potrebné sledovať. Zdrojový kód je možné preložiť pomocou projektu `SigantureTest.sln`. Vo vrchnej zložke sa tiež nachádza preložený spustiteľný súbor finálnej verzie tohto testera `SignatureTest.exe`, ktorého analýza je používaná na testovanie skriptom z adresára `testscript`.

C.2.4 Testovací skript na overenie funkcionalít analyzátora

Testovací skript dostupný v adresári `testscript` slúži na automatickú kontrolu stavu analyzátora po analýze testovacieho programu opísaného v predchádzajúcej časti. Jeho úlohou je pomocou parametru, ktorým je cesta k nespracovaným dátam analýzy, realizovať lokálnu analýzu presne tak, ako prebieha v Cuckoo. Na to používa nástroje z modulov *Processing* a *Signature* sandboxu, ktoré však nie sú súčasťou prílohy. Následne, po spustení všetkých signatúr nad dátami z behu testera, kontroluje stav analyzátora pomocou jednoduchých funkcií `assert` štandardnej knižnice Python `unittest`. V prípade chybného parametru niektorého z vnútorných objektov analyzátora nastane chyba, ktorá je oznámená vo výstupe.

C.2.5 Zdrojový kód analyzátora

Kód analyzátora je celý obsiahnutý v súbore `crypto_api_analysis.py` a väčšina konštánt je oddelená pre prehľadnosť v súbore `crypto_api_utils\constants.py`.

Hlavná trieda je trieda `CryptoAnalyser`, ktorá vykonáva všetky úkony potrebné pri každom volaní. Metóda `on_call` tejto triedy spracováva všetky sledované CryptoAPI volania. V rámci Cuckoo však musí byť objekt tejto triedy integrovaný pomocou tried dediacich z triedy Cuckoo `Signature`. Hlavnou triedou `Signature` je trieda `CryptoAnalysisSignature`, ktorá spracováva všetky vybrané volania CryptoAPI a predáva ich ďalej na analýzu zdieľanému objektu `g_crypto_analyser` typu `CryptoAnalysisSignature`. Ostatné triedy `Signature` plnia doplnkové funkcie, ako boli v častiach 4 a 5 opísané. Nepoužívajú metódu `on_call` ale sú púšťané len raz volaním metódy `run` (opísané v časti 1.1.3). Signatúra `CryptoExceptionSignature` slúži na oznámenie nezachytenej výnimky v signatúre `CryptoAnalysisSignature`. Signatúra `CryptoFlaggingSignature` slúži na filtrovanie detekcií hlavnej signatúry pre prípady, že nastala aktivita CryptoAPI len v systémovom procese spustenom analyzovaným programom.